arXiv:2304.02327v1 [math.NA] 5 Apr 2023

克罗内克形式下用于 d 维问题的指数积分器中 φ 函数的方向分裂

Marco Caliari* Fabio Cassini†

2023年4月6日

摘要

本文提出了一种高效、实用且易于实现的方法，用于近似计算指数积分器（最高至二阶）中具有 d 维克罗内克和结构的矩阵上的 φ 函数作用。该方法基于对所涉及矩阵函数的方向分裂，从而能够以 μ 模式利用高效的三级 BLAS，实际计算所需的函数作用。我们在二维和三维问题上，使用多种指数积分器成功测试了该方法；与一种专为计算克罗内克和的 φ 函数作用而设计的技术相比，该方法始终能显著加速计算。

关键词：指数积分器， μ 模式， φ 函数，方向分裂，克罗内克形式

1 引言

近年来，计算具有 Kronecker 和结构的指数函数及类指数函数作用的问题受到了大量关注 [6, 8, 10, 13, 20, 21, 24]。事实上，对此类量的高效逼近使得我们能够有效运用指数积分器，对大型刚性常微分方程（ODE）组进行时间积分。更具体地说，我们假设所处理的是如下刚性常微分方程组

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(t, \boldsymbol{u}(t)), \quad t > 0, \\ \boldsymbol{u}(0) = \boldsymbol{u}_0. \end{cases} \tag{1a}$$

这里， $\boldsymbol{g}(t, \boldsymbol{u}(t))$ 是关于 t 和未知量 $\boldsymbol{u}(t) \in \mathbb{C}^N$ 的一般非线性函数，其中 $N = n_1 \cdots n_d$，而 $K \in \mathbb{C}^{N \times N}$ 是一个具有 d 维 Kronecker 和结构的矩阵，即，

$$K = A_d \oplus A_{d-1} \oplus \cdots \oplus A_1 = \sum_{\mu=1}^{d} A_{\otimes\mu}, \quad A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1, \tag{1b}$$

其中 $A_\mu \in \mathbb{C}^{n_\mu \times n_\mu}$，而 I_μ 是大小为 n_μ 的单位矩阵。本文中，符号 $\otimes$ 表示矩阵的标准 Kronecker 积，而 $\oplus$ 表示矩阵的 Kronecker 和。此外，我们将系统 (1) 称为克罗内克形式系统，或具有 Kronecker 和结构的系统。

*意大利维罗纳大学计算机科学系， marco.caliari@univr.it，通讯作者
†意大利维罗纳大学计算机科学系， fabio.cassini@univr.it

这类系统自然出现在许多领域。例如，对于 $d = 2$，这种结构会出现在常系数矩阵 Riccati 微分方程中（例如，参见文献 [1, 第 3] 章）

$$\begin{cases} \boldsymbol{U}'(t) = A_1\boldsymbol{U}(t) + \boldsymbol{U}(t)A_2^{\mathsf{T}} + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{U}_0, \end{cases} \tag{2}$$

其中 $\boldsymbol{U}(t) \in \mathbb{C}^{n_1\times n_2}$、 $B \in \mathbb{C}^{n_2\times n_1}$ 和 $C \in \mathbb{C}^{n_1\times n_2}$。事实上，利用克罗内克乘积 [10], 的性质，我们可以将这样一个矩阵方程等价地重写为克罗内克形式（1）中的常微分方程组，即

$$\begin{cases} \boldsymbol{u}'(t) = ((I_2 \otimes A_1) + (A_2 \otimes I_1))\boldsymbol{u}(t) + \operatorname{vec}(C + \boldsymbol{U}(t)B\boldsymbol{U}(t)), \\ \boldsymbol{u}(0) = \operatorname{vec}(\boldsymbol{U}_0), \end{cases} \tag{3}$$

其中 vec 是将输入矩阵的各列堆叠成单个向量的算子。

具有Kronecker和结构的系统也常常出现在使用线法，对定义在张量积区域上并满足适当边界条件的偏微分方程（PDE）进行数值求解时。事实上，对 Allen–Cahn、Brusselator、Gray–Scott、对流–扩散–反应 [8, 10] 或薛¨定谔方程 [6], 等著名抛物型方程进行空间半离散化后，我们得到一个形式为（1）的大型刚性常微分方程组。

给定系统 (1) 后，可以采用许多技术对其进行数值时间积分，我们尤其关注指数积分器 [18] 的应用。事实上，指数积分器是完成这项任务的一种重要方法，因为它们具有良好的稳定性，适用于刚性问题。这类方法需要计算矩阵指数以及类指数矩阵函数（即所谓的 φ-函数）作用于向量的结果。对于一般矩阵 $X \in \mathbb{C}^{N\times N}$，它们定义如下:

$$\varphi_0(X) = \mathrm{e}^X, \quad \varphi_\ell(X) = \frac{1}{(\ell-1)!}\int_0^1 \mathrm{e}^{(1-\theta)X}\theta^{\ell-1}d\theta, \quad \ell > 0, \tag{4a}$$

其泰勒级数展开为

$$\varphi_\ell(X) = \sum_{i=0}^{\infty}\frac{X^i}{(i+\ell)!}, \quad \ell \geq 0. \tag{4b}$$

当 X 的规模允许时，实践中通常使用对角 Pad´e 逼近 [2, 5, 29] 或多项式逼近 [19]来近似此类矩阵函数。另一方面，当 X 规模较大时，这种方法在计算上难以实现，因此人们开发了许多算法，直接计算 φ-函数作用于向量的结果。我们提及其中的若干种方法：基于 Krylov 子空间的技术 [15, 22, 25], 直接多项式方法 [3,7, 11, 19], 以及混合技术 [9]。事实上，当 X 是具有 Kronecker 和结构 (1b) 的矩阵 K 时，可以利用这一信息，更高效地计算 φ-函数作用于向量的结果。具体而言，考虑 $\ell = 0$，使得 $\varphi_0(K) = \mathrm{e}^K$。于是容易看出，计算

$$\boldsymbol{e} = \mathrm{e}^K\boldsymbol{v} = \mathrm{e}^{A_d\oplus A_{d-1}\oplus\cdots\oplus A_1}\boldsymbol{v} = \left(\mathrm{e}^{A_d}\otimes\mathrm{e}^{A_{d-1}}\otimes\cdots\otimes\mathrm{e}^{A_1}\right)\boldsymbol{v} \tag{5}$$

在数学上等价于计算

$$\boldsymbol{E} = \boldsymbol{V}\times_1\mathrm{e}^{A_1}\times_2\cdots\times_d\mathrm{e}^{A_d}, \tag{6}$$

我们将其称为张量形式。这里， $\boldsymbol{E}$和 $\boldsymbol{V}$是大小为 $n_1 \times \cdots \times n_d$、阶数为 d的张量，满足 vec($\boldsymbol{E}$) $= \boldsymbol{e}$ 和 vec($\boldsymbol{V}$) $= \boldsymbol{v}$，其中 vec 是将输入张量的各列堆叠成适当的单列向量的算子。符号 $\times_\mu$ 表示沿模式 μ的张量–矩阵乘积，也称为 μ-模乘积；连续执行 μ-模乘积（如公式 (6) 所示）的计算通常称为 Tucker 算子。注意，张量 $\boldsymbol{E}$ 的元素 $e_{i_1 \ldots i_d}$ 结果为

$$e_{i_1 \ldots i_d} = \sum_{j_d=1}^{n_d} \cdots \sum_{j_1=1}^{n_1} v_{j_1 \ldots j_d} \prod_{\mu=1}^{d} e^{\mu}_{i_\mu j_\mu}, \quad 1 \le i_\mu \le n_\mu, \tag{7}$$

其中 $e^{\mu}_{i_\mu j_\mu}$ 是 e^{A_μ}的一般元素。尽管公式 (5)、(6)和 (7)在数学上是等价的，但直接使用公式 (5) 和(7) 的效率远低于公式 (6)；公式(6)在计算小规模矩阵指数 e^{A_μ} 后，通过利用高性能的第3级 BLAS来实现。事实上，例如，在二维情况下，公式(6)需要进行两次矩阵–矩阵乘法，因为它可化简为$\mathrm{e}^{A_1} \boldsymbol{V} \left(\mathrm{e}^{A_2}\right)^{\mathsf{T}}$；而在 d 维情况下，它需要 d 次第 3 级BLAS调用。这项技术促成了所谓的 μ-模式积分器 [6], ，并已成功用于对流–扩散–反应方程和 Schr¨odinger 方程的半离散化进行时间积分，最终还可与分裂格式结合使用。特别是，据报道，与用于计算矩阵指数作用于向量的最先进技术相比，该方法具有显著的加速效果，并且在使用图形处理器进行数值模拟时表现出非常良好的扩展性。我们建议有兴趣进一步了解 Tucker 算子及其应用的读者查阅参考文献 [6, 10]。

在计算高阶 φ 函数的作用时，即 $\varphi_\ell(K)\boldsymbol{v}$ （其中 $\ell > 0$），公式 (5) 中的最后一个等式不再成立。在参考文献 [8] 中，作者提出了一种克服这一困难的方法：基于将求积公式应用于 φ 函数的积分定义 (4a)，该方法要求在每个求积点上，由 Tucker 算子对张量 $\boldsymbol{V}$ 执行矩阵指数的作用。这样，就可以在给定容差下计算所需的 φ 函数作用。这项被命名为 PHIKS 的技术已针对任意维数 d 开发，其设计不仅用于计算作用于向量的 φ 函数，还可计算 φ 函数作用的线性组合。此外，所需量可在适当的不同时间尺度上同时获得。这些特性使得实现高刚性阶指数积分器（如指数 Runge–Kutta 格式）比使用当前最先进技术计算 φ 函数作用的组合更为高效。另一种基于将求积法则应用于公式 (4a) 的最新方法见参考文献 [13], ，其中该技术仅在维数 $d \le 3$ 下针对给定时间尺度上单个 φ 函数的作用进行了描述。文献中还有其他针对具有 Kronecker 和结构的矩阵计算 φ 函数作用的方法。例如，参考文献 [24], 中的算法基于 Sylvester 方程的求解，目前仅限于维数 $d = 2$。另一种在可能涉及低秩近似的情形下，近似 Sylvester 算子 $A_1\boldsymbol{V} + \boldsymbol{V}A_2^{\mathsf{T}}$ 或 Lyapunov 算子 $A\boldsymbol{V} + \boldsymbol{V}A^{\mathsf{T}}$ 的 φ 函数作用以求解 Riccati 微分方程的方法，见参考文献 [20, 21]。

本文提出了一种替代方法，用于近似 $\varphi_\ell(K)\boldsymbol{v}$，其中 $\ell > 0$ 和 K是具有 d维 Kronecker 和结构的矩阵，适用于二阶及以下的指数积分器。该方法称为 PHISPLIT，基于对 K 的矩阵 φ-函数进行方向分裂，由此产生的近似误差与时间推进数值格式的误差相当。所需作用的计算采用 μ-模式方式，通过每个 φ-函数对应的单个 Tucker 算子完成，并利用高效的 3 级 BLAS。在回顾第2节中的一些常用指数积分器后，

我们将在第3节介绍所提出的技术，以及如何将其用于实现前文提到的指数格式。随后，在第4节展示一些数值实验，以说明 PHISPLIT 的有效性，最后在第5节总结全文。

2 回顾一些二阶以内的指数积分器

当对形式为(1)的刚性半线性常微分方程进行数值积分时，其中刚性部分由矩阵 K表示，一种重要的方法是使用指数积分器 [18]。为方便读者，本文在此给出将在第4节数值实验中使用的指数格式的一种可能推导（为简单起见，假设时间步长为常数）。

出发点是常数变易公式

$$\begin{aligned}\boldsymbol{u}(t_{n+1}) &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \mathrm{e}^{(t_{n+1}-s)K}\boldsymbol{g}(s,\boldsymbol{u}(s))ds \\ &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))d\theta\end{aligned} \tag{8}$$

该式表示系统 (1a) 在时刻 $t_{n+1} = t_n + \tau$ 的解析解，其中 τ为时间步长。若用矩形左规则近似该积分，则得到该格式

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}_n) = \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \tag{9}$$

这被称为 Lawson–Euler 格式（参见参考文献 [4, A.1 节1]）。它是一个一阶格式，对于具有常系数的线性齐次问题具有精确性。系统 (1a) 的线性部分可以精确求解，因此无需因刚性而对时间步施加任何限制。如果改为对方程 (8) 中的积分应用梯形求积公式，则可得到如下近似:

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \frac{\tau}{2}\left(\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)) + \boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1}))\right).$$

随后，通过使用Lawson–Euler格式(9)创建一个中间阶段 $\boldsymbol{u}_{n2}$ ，在右端项中近似 $\boldsymbol{u}(t_{n+1})$，即可得到一个显式时间推进格式。总体而言，我们得到

$$\begin{aligned}\boldsymbol{u}_{n2} &= \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \\ \boldsymbol{u}_{n+1} &= \mathrm{e}^{\tau K}\left(\boldsymbol{u}_n + \frac{\tau}{2}\boldsymbol{g}(t_n,\boldsymbol{u}_n)\right) + \frac{\tau}{2}\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}),\end{aligned} \tag{10}$$

which是一种二阶Lawson方法，在文献中也称为 Lawson2b（参见参考文献 [4, Sec. A.1.6]）.

对公式 (8) 中积分的近似采用另一种方法，可以得到所谓的指数Runge–Kutta方法。实际上，如果我们仅将非线性函数$\boldsymbol{g}(t_n+\tau\theta, \boldsymbol{u}(t_n+\tau\theta))$近似为 $\boldsymbol{g}(t_n, \boldsymbol{u}(t_n))$，利用方程 (4a) 中 φ_1函数的定义，我们得到如下格式

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{g}(t_n,\boldsymbol{u}_n),$$

其等价地可改写为

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)) \tag{11}$$

这被称为指数欧拉法（或指数 Nørsett–Euler，参见参考文献 [4, Sec. A.2.1]）。这是一阶格式，对于常系数线性问题是精确的。另一种

可能性是用关于 θ 的一次多项式，在 0 和 1 处对 $\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))$ 进行插值，从而得到近似

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}(\theta\boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1})) + (1-\theta)\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)))d\theta.$$

在右端项中通过指数欧拉法取一个阶段值 $\boldsymbol{u}_{n2}$，以近似 $\boldsymbol{u}(t_{n+1})$，并利用公式 (4a) 中 φ_1 和 φ_2 函数的定义，可得到二阶指数龙格–库塔格式（文献中也称为 ETD2RK，参见参考文献 [4,第 A.2 节5]）

$$\begin{aligned}\boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)),\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_{n2} + \tau\varphi_2(\tau K)(\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}) - \boldsymbol{g}(t_n,\boldsymbol{u}_n)).\end{aligned} \tag{12}$$

最后，我们考虑罗森布洛克–欧拉方法（参见参考文献 [18, 例 2.20]）。在自治情形下，该方法可通过将指数Euler格式应用于线性化微分方程得到

$$\boldsymbol{u}'(t) = \left(K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)\right)\boldsymbol{u}(t) + \left(\boldsymbol{g}(\boldsymbol{u}(t)) - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)\boldsymbol{u}(t)\right) = K_n\boldsymbol{u}(t) + \boldsymbol{g}_n(\boldsymbol{u}(t)),$$

wh其中 K_n 是取值于 $\boldsymbol{u}_n$ 处的雅可比矩阵，$\boldsymbol{g}_n(\boldsymbol{u}(t))$ 是余项。由此得到的格式 i s

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K_n)(K\boldsymbol{u}_n + \boldsymbol{g}(\boldsymbol{u}_n)). \tag{13}$$

这是一种二阶方法，与上述所有方法不同，它在每个时间步都需要计算一个不同的矩阵函数 $\varphi_1(\tau K_n)$。推广到非自治系统是直接的，参见参考文献 [18, 例 2.21]。

注1. 我们在此仅考虑了少数需要作用于 φ-函数的指数积分器。其他一阶或二阶指数型格式也可以受益于 μ-模式分裂技术，用于计算我们在本文中提出的克罗内克和的 φ-函数。其中包括校正分裂格式 *[14],* 低正则性格式 *[27],* 以及针对线性时变系数非齐次方程的 Magnus 积分器 *[16]*。

3 φ-函数的方向分裂

如引言中所述，我们假设所处理的是一个具有 Kronecker 和结构 (1b) 的矩阵 K，并且我们希望在指数积分器的应用背景下，高效地计算 $\varphi_\ell(\tau K)\boldsymbol{v}$, $\boldsymbol{v}\in\mathbb{C}^N$, $\tau\in\mathbb{R}$。特别地，我们知道，通过采用一个收敛阶为 p 的格式，可以使局部误差达到 $\mathcal{O}(\tau^{p+1})$，其中 τ 为（常数）时间步长。因此，如果积分器需要计算形式为 $\tau^q\varphi_\ell(\tau K)$ 的量，其中 $q>0$，那么只需以误差 $\mathcal{O}(\tau^{p+1-q})$ 对 $\varphi_\ell(\tau K)$ 进行近似，即可保持原有的收敛阶。对于我们所关注的格式，即上一节中介绍的格式，我们将采用以下结果。

定理 1。设 K 是一个具有 d维Kronecker 和结构 (1b) 的矩阵。则，对于$\ell>0$，我们有

$$\varphi_\ell(\tau K) = (\ell!)^{d-1}\left(\varphi_\ell(\tau A_d)\otimes\varphi_\ell(\tau A_{d-1})\otimes\cdots\otimes\varphi_\ell(\tau A_1))\right) + \mathcal{O}(\tau^2). \tag{14}$$

证明。为使表述简洁，我们采用以下记号

$$X_d \otimes X_{d-1} \otimes \cdots \otimes X_1 = \bigotimes_{\mu=d}^{1} X_\mu, \quad X_\mu \in \mathbb{C}^{n_\mu \times n_\mu}.$$

然后，利用 φ_ℓ 函数 (4b)的泰勒展开以及Kronecker 积的性质（有关全面综述请参见参考文献 [30] ），我们得到

$$\begin{aligned}
(\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \varphi_\ell(\tau A_\mu) &= (\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \left(\frac{I_\mu}{\ell!} + \frac{\tau A_\mu}{(\ell+1)!} + \mathcal{O}(\tau^2) \right) \\
&= (\ell!)^{d-1} \left(\frac{1}{(\ell!)^d} \bigotimes_{\mu=d}^{1} I_\mu + \frac{\tau}{(\ell!)^{d-1}(\ell+1)!} \sum_{\mu=1}^{d} A_{\otimes\mu} + \mathcal{O}(\tau^2) \right) \\
&= \frac{I}{\ell!} + \frac{\tau K}{(\ell+1)!} + \mathcal{O}(\tau^2) \\
&= \varphi_\ell(\tau K) + \mathcal{O}(\tau^2),
\end{aligned}$$

其中 I 是大小为 $N \times N$ 的单位矩阵。 □

公式(14)允许采用高效的基于 μ 模态的实现方式，类似于矩阵指数情形(6)。事实上，给定一个阶-d 张量 $\boldsymbol{V}$，使得 $\boldsymbol{v} = \text{vec}(\boldsymbol{V})$，如果我们定义

$$\boldsymbol{p}_\ell^{(2)} = \varphi_\ell^{(2)}(\tau K)\boldsymbol{v} = \text{vec}\left(((\ell!)^{d-1}\boldsymbol{V}) \times_1 \varphi_\ell(\tau A_1) \times_2 \varphi_\ell(\tau A_2) \times_3 \cdots \times_d \varphi_\ell(\tau A_d) \right)$$

我们有

$$\varphi_\ell(\tau K)\boldsymbol{v} = \boldsymbol{p}_\ell^{(2)} + \mathcal{O}(\tau^2).$$

我们将其称为

$$\boldsymbol{P}_\ell^{(2)} = \left((\ell!)^{d-1}\boldsymbol{V} \right) \times_1 \varphi_\ell(\tau A_1) \times_2 \varphi_\ell(\tau A_2) \times_3 \cdots \times_d \varphi_\ell(\tau A_d)$$

作为 $\boldsymbol{p}_\ell^{(2)}$ 的张量形式，因此 $\text{vec}(\boldsymbol{P}_\ell^{(2)}) = \boldsymbol{p}_\ell^{(2)}$。这正是我们提议采用的形式，用于计算具有Kronecker和结构的矩阵的 φ-函数在上述指数积分器中的作用。从现在起，我们将这一技术称为 PHISPLIT 方法。请注意，在计算小型矩阵函数$\varphi_\ell(\tau A_\mu)$（其中 $\mu = 1, .., d$）之后，只需一个 Tucker 算子即可计算张量形式近似值。

3.1 小规模矩阵 φ-函数的计算

矩阵 A_μ 的尺寸比 K小得多，相应的矩阵 φ-函数可以轻松地直接计算。具体而言，对于 φ_0（即指数函数），我们采用适用于一般矩阵的最常用技术，该技术基于对角有理 Padé 逼近，并结合缩放平方算法（参见参考文献 [2]）。这一过程封装在内部 MATLAB 函数 expm 中。其他也可使用的算法基于矩阵指数的泰勒逼近，详见参考文献 [12, 28]。

对于高阶矩阵 φ 函数的计算，我们依赖于应用于积分定义式 (4a) 的求积公式。对于一般矩阵 $X \in \mathbb{C}^{N\times N}$，于是有

$$\varphi_\ell(X) \approx \sum_{i=1}^{q} w_i \mathrm{e}^{(1-\theta_i)X} \frac{\theta_i^{\ell-1}}{(\ell-1)!}, \quad \ell > 0. \tag{15}$$

为了避免求积点数量大到不切实际，我们将该过程与改进的缩放与平方算法结合起来（参见参考文献 [29]）。具体而言，我们将原矩阵 X 缩放 2^s 倍，其中 s 是一个自然数，其定义满足 $\|X/2^s\|_1 < 1$；我们利用公式 (15) 近似计算 $\varphi_\ell(X/2^s)$，再通过递推关系恢复 $\varphi_\ell(X)$

$$\varphi_\ell(2z) = \frac{1}{2^\ell}\left[\mathrm{e}^z\varphi_\ell(z) + \sum_{k=1}^{\ell}\frac{\varphi_k(z)}{(\ell-k)!}\right]. \tag{16}$$

为了计算所需的矩阵指数，我们再次使用内部 MATLAB函数 expm。请注意，对 φ_ℓ 进行平方也需要计算所有 φ_j 函数，其中 $0 < j < \ell$。具体而言，在第一次平方时，我们使用公式 (15) 并结合 $\ell = j$ 来计算这些函数，同时使用求积过程 $\varphi_\ell(X/2^s)$中已得到的同一组矩阵指数。对于之后的每一步平方，我们直接使用公式 (16)，并结合$\ell = j$。因此，计算单个矩阵函数 φ_ℓ即可得到所有矩阵函数 φ_j，其中 $0 \le j \le \ell$。实际计算中，我们采用高斯–勒让德–洛巴托公式进行求积，这样只需适量的求积节点即可达到高精度。此外，由于该公式使用求积区间的端点，我们会用到矩阵指数 $\mathrm{e}^{X/2^s}$ ($\theta_1 = 0$)；它也用于后续的平方过程，因此我们无需生成最后一个矩阵指数，因为 $\theta_q = 1$。整个过程通过我们的函数 phiquad 使用 MATLAB 语言实现。

或者，对于矩阵 φ-函数的计算，可以使用MATLAB 例程 phipade（参见参考文献 [5]），其算法基于有理Padé 逼近，并结合平方公式（16）。参考文献 [19]介绍了另一种近期技术，该技术采用多项式泰勒逼近而非有理 Padé 逼近。

3.2 指数积分器的实际实现

第 2 节中介绍的 Lawson 方法只需要进行矩阵指数的作用，其实现不会受到任何方向分裂误差的影响，这得益于公式 (5)和 (6) 之间的等价性。特别地，Lawson–Euler 的张量形式是

$$\boldsymbol{U}_{n+1} = (\boldsymbol{U}_n + \tau\boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d}, \tag{17}$$

而 Lawson2b格式表示为

$$\begin{aligned}\boldsymbol{U}_{n2} &= (\boldsymbol{U}_n + \tau\boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d},\\ \boldsymbol{U}_{n+1} &= \Big(\boldsymbol{U}_n + \frac{\tau}{2}\boldsymbol{G}(t_n, \boldsymbol{U}_n)\Big) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \frac{\tau}{2}\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}).\end{aligned} \tag{18}$$

在此以及后续公式中，我们有

$$\mathrm{vec}(\boldsymbol{U}) = \boldsymbol{u}, \quad \mathrm{vec}(\boldsymbol{G}(t, \boldsymbol{U})) = \boldsymbol{g}(t, \boldsymbol{u}).$$

The 剩余的指数积分器转换如下。首先，矩阵的作用

$\mathrm{n}^{\mathrm{x}K}{}_{\mathrm{o}}\boldsymbol{u}_n$ 以张量形式计算为

$$\sum_{\mu=1}^{d}(\boldsymbol{U}_n \times_\mu A_\mu)$$

无需显式组装矩阵 K （参见参考文献 [9]）。指数欧拉法 PHISPLIT方法为

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \tau\left(\sum_{\mu=1}^{d}(\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(t_n, \boldsymbol{U}_n)\right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d). \tag{19}$$

请注意，对于 $\boldsymbol{G}(t_n, \boldsymbol{U}_n)$ 的贡献特别小的情况，另一种更合适的形式为

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d),$$

事实上，这是线性自治问题的精确公式。通过考察指数欧拉方法的公式 (19)，ETD2RK PHISPLIT 格式可写为

$$\begin{aligned} \boldsymbol{U}_{n2} &= \boldsymbol{U}_n + \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d), \\ \boldsymbol{U}_{n+1} &= \boldsymbol{U}_{n2} + \tau \left(\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}) - \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_2(\tau A_1) \times_2 \cdots \times_d \varphi_2(\tau A_d). \end{aligned} \tag{20}$$

最后，对于自治系统的指数 Rosenbrock–Euler 方法，我们假设雅可比矩阵 K_n 可以写成 Kronecker 和，即

$$K_n = K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n) = J_d(\boldsymbol{U}_n) \oplus J_{d-1}(\boldsymbol{U}_n) \oplus \cdots \oplus J_1(\boldsymbol{U}_n).$$

因此，指数 Rosenbrock–Euler PHISPLIT 方法为

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \tau \boldsymbol{G}(\boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau J_1(\boldsymbol{U}_n)) \times_2 \cdots \times_d \varphi_1(\tau J_d(\boldsymbol{U}_n)). \tag{21}$$

4 数值实验

本节通过数值实验验证所提出的方法 PHISPLIT。具体而言，我们将考虑一个来自线性二次控制的二维示例，以及一个模拟对流–扩散–反应方程的三维示例。为进行时间推进，我们将采用第3节.2中所述的第2节中的指数积分器，使用 PHISPLIT 版本。

作为比较，我们将考虑使用 PHIKS[1] [8]对具有 Kronecker 和结构的矩阵上的 φ-函数作用进行近似。该算法同样采用张量形式和 μ-模乘积，但需要输入容差；我们将其设为与方法的局部时间阶和当前解的范数成正比。比例常数的选取应确保该例程产生的误差（相对于参考解或解析解测量）不会影响时间离散误差。

为计算所有相关的张量运算，即 Tucker 算子和 μ 模乘积，我们使用 KronPACK[2] 软件包中的函数。此外，为了计算所需的矩阵 φ-函数，我们采用内部函数 expm（用于 φ_0）和函数 phiquad（用于 φ_ℓ, $\ell > 0$），如第3节 3.1 所述。在硬件方面，所有实验均使用配备六个物理核心和 16GB RAM 的 Intel® Core™ i7-10750H 中央处理器运行。软件方面，我们使用 MathWorks MATLAB® R2022a。用于复现数值示例的所有代码，以及我们对 PHISPLIT 的实现和函数 phiquad，均可在一个持续维护的 GitHub 代码仓库[3]中找到。最后需要强调的是，函数 phisplit 和 phiquad 与 GNU Octave 完全兼容。

[1] https://github.com/caliarim/phiks
[2] https://github.com/caliarim/KronPACK
[3] https://github.com/caliarim/phisplit

4.1 线性二次控制

本节展示一个线性二次控制中的经典示例（例如，参见参考文献 [23, 26]）。我们关注对泛函进行最小化，其中标量控制 $v(t) \in \mathbb{R}$ 为

$$\mathcal{J}(v) = \frac{1}{2}\int_0^T \left(\alpha s(t)^2 + v(t)^2\right) dt$$

满足以下约束

$$\begin{aligned} &w'(t) = Aw(t) + bv(t), \quad w(0) = w_0, \\ &s(t) = cw(t). \end{aligned}$$

这里， $w(t) \in \mathbb{R}^{n\times 1}$ 是包含状态变量的列向量， $s(t) \in \mathbb{R}$ 表示标量输出， $A \in \mathbb{R}^{n\times n}$ 是系统矩阵， $b \in \mathbb{R}^{n\times 1}$ 是系统列向量， $c \in \mathbb{R}^{1\times n}$是行向量， $\alpha \in \mathbb{R}^+$ 是正标量。

因此，约束优化问题的解由最优值确定 1
control

$$v^*(t) = -b^{\mathsf{T}}\boldsymbol{U}(t)w(t),$$

其中， $\boldsymbol{U}(t) \in \mathbb{R}^{n\times n}$ 满足对称 Riccati 微分方程

$$\begin{cases} \boldsymbol{U}'(t) = A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{Z}, \end{cases} \tag{22}$$

其中， $C = \alpha c^{\mathsf{T}}c$ 和 $B = -bb^{\mathsf{T}}$ （有关该主题的全面介绍，请参阅参考文献 [1, 第 4] 章）。

这里 $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ 是一个元素全为零的矩阵。显然，方程(22)具有形式(2)，进而可视为具有二维 Kronecker 和结构(3)的问题，并可借助第3节所述技术高效积分。还需注意，方程(22)的解收敛到由代数 Riccati 方程确定的稳态

$$A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t) = 0. \tag{23}$$

在我们的数值实验中，与文献中先前的做法类似 [20, 21, 23,26], ，我们将 $A \in \mathbb{R}^{\hat{n}^2\times\hat{n}^2}$ 取为对算子进行二阶中心有限差分离散化所得的矩阵

$$\partial_{xx} + \partial_{yy} - 10x\partial_x - 100y\partial_y \tag{24}$$

在具有齐次 Dirichlet 边界条件的区域 $[0,1]^2$ 上。此外，向量 b 的分量 b_k 定义为

$$b_k = \begin{cases} 1 \text{ if } 0.1 < x_i \le 0.3, \\ 0 \text{ otherwise,} \end{cases} \quad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n},$$

而对于向量 c 的分量 c_k ，我们取

$$c_k = \begin{cases} 1 \text{ if } 0.7 < x_i \le 0.9, \\ 0 \text{ otherwise,} \end{cases} \quad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n}.$$

这里， x_i 表示沿 x 方向的第 i个（内部）网格点。最后，我们设定 $\alpha = 100$。

对于方程 (22) 的时间积分，我们采用指数Rosenbrock–Euler 方法，该方法已在参考文献 [20, 21], 中使用，并在公式(13) 中进行了报告（有关 PHISPLIT 版本，请参见公式 (21)）。事实上，系统 (22) 的雅可比矩阵具有以下 Kronecker 和结构

$$K_n = I \otimes (A^\mathsf{T} + \boldsymbol{U}_n B) + (A + B\boldsymbol{U}_n)^\mathsf{T} \otimes I,$$

其中 I 是大小为 $n \times n$ 的单位矩阵，且 $n = \hat{n}^2$。我们指出，指数 Rosenbrock–Euler PHISPLIT 方法在每个时间步都需要计算矩阵函数$\varphi_1(\tau(A^\mathsf{T} + \boldsymbol{U}_n B))$、计算作用 $K\boldsymbol{u}_n$并执行一次 Tucker 算子。我们还将采用二阶指数 Runge–Kutta 方法 ETD2RK，该方法见公式 (12)，并以 PHISPLIT 意义在公式 (20) 中给出。尽管该积分器的每个时间步对 PHISPLIT 版本需要两个 Tucker 算子以及作用 $K\boldsymbol{u}_n$，但在恒定时间步长实现中，所需的矩阵函数 $\varphi_1(\tau A^\mathsf{T})$ 和 $\varphi_2(\tau A^\mathsf{T})$ 可以在开始时一次性全部计算完成。

首先，我们验证所涉及的指数积分器在长期仿真中的实现，即直至达到稳态。在本实验中，我们对 x 和 y 变量采用 $\hat{n} = 20$个内部离散点。如图 1 中的曲线所证实，我们看到所有方法，无论是其 PHIKS 还是 PHISPLIT 实现，都在时间约 0.15 时达到方程 (23) 的解，该解由控制系统工具箱中的 MATLAB 函数 icare 得到。

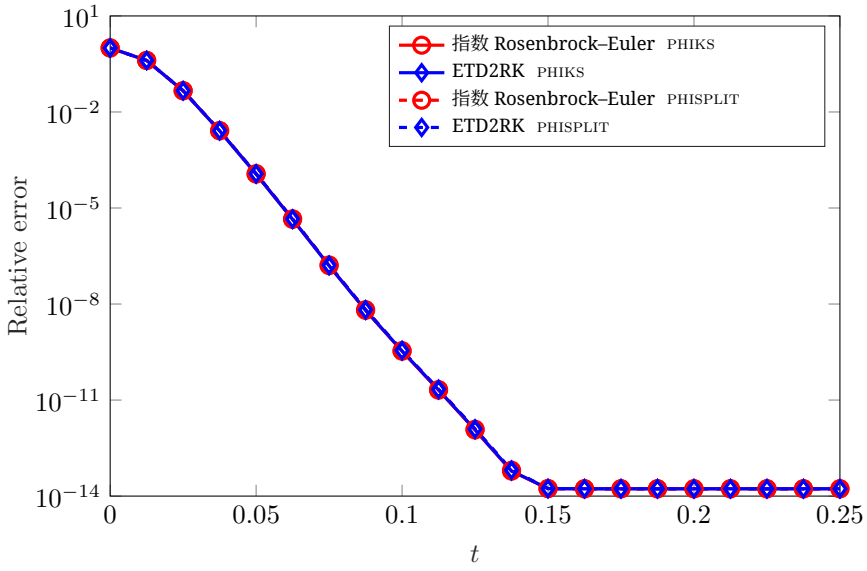


图 1：指数 Rosenbrock–Euler 方法和ETD2RK方法在 PHIKS 和 PHISPLIT 变体中收敛至 Riccati 微分方程 (22) 的稳态。此处 $\hat{n} = 20$，积分器使用了 200 个时间步，并显示了相对于代数 Riccati方程(23) 的解、以 Frobenius 范数测量的相对误差，每 10 个时间步显示一次。

接下来，我们比较各积分器在求解方程（22）时的性能，其中参数为 $\hat{n} = 30$，终止时间为 $T = 0.025$。所有方法均采用不同的时间步长运行，使其相对于参考解的相对误差大致相当。各方法所用的时间步

数以及每次数值模拟中数值观测到的收敛阶列于表 1。正如预期，所有方法均表现为二阶方法。

指数 Rosenbrock–Euler PHIKS	步骤	10	20	30	40	50
	阶数	–	2.11	2.06	2.05	2.03
ETD2RK PHIKS	步骤	7	14	21	28	35
	阶	–	2.08	2.05	2.03	2.03
指数 Rosenbrock–Euler PHISPLIT	步	30	65	100	135	170
	阶	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	步	30	65	100	135	170
	阶	–	2.05	2.03	2.02	2.02

表1：使用不同的指数积分器和 $\hat{}n = 30$ 对 Riccati 微分方程(22)进行时间积分直至最终时间 $T = 0.025$时的时间步数和观测到的收敛率。所达到的误差和墙钟时间显示于图 2 中。

在图2中，我们报告了模拟的相对误差及其对应的墙钟时间。在此，我们还纳入了内置 MATLAB 函数 ode23的性能。这是一种三阶显式 Runge–Kutta方法，采用变步长，适用于要求不严的容差以及中度刚性问题。事实上，事实证明，在达到与其他方法相近范围内的精度方面，它是常微分方程求解器套件中最快的例程。首先，我们注意到，在 PHIKS 实现中，指数 Rosenbrock–Euler 方法始终比ETD2RK 更快，也就是说，矩阵函数的作用是在不会影响时间离散误差的精度下计算的（参见本节开头的讨论）。另一方面，采用 PHISPLIT 的两种实现始终比其PHIKS 对应实现更快，尽管它们需要更多的时间步才能达到可比的精度。此外，相对于指数 Rosenbrock–Euler 方法，ETD2RK方法的速度更快。这主要是因为，在 Runge–Kutta情形下，矩阵函数只需在时间推进之前计算一次。事实上，该方法至少比其他指数方法快一倍，并且比ode23 更快；不过，对于最严格的容差，ode23 仍表现良好。

最后，我们使用 $\hat{}n = 40$重复了相同的实验。结果见表 2 和图3。与前一种情况相比，总体表现相似，尽管 PHISPLIT 实现 相对于其 PHIKS 对应实现 的加速比明显更大。事实上，ETD2RK PHISPLIT 仍然是性能最佳的方法。

注 2. 算子 (24) 的离散化本身具有 Kronecker 和结构。因此，可以将方程 (22)（为简化表述，采用向量形式）写作

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u} + \boldsymbol{g}(\boldsymbol{u}), \\ \boldsymbol{u}(0) = \boldsymbol{z}, \end{cases}$$

其中， $\boldsymbol{g}$ 和 $\boldsymbol{z}$ 分别是非线性项和 $\boldsymbol{Z}$ 的向量化，而 K 具有如下形式

$$K = I \otimes I \otimes I \otimes D_1^{\mathsf{T}} + D_2^{\mathsf{T}} \otimes I \otimes I \otimes I.$$

这里， I 是大小为 $\hat{}n \times \hat{}n$ 的单位矩阵，而 $D_1 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ 和 $D_2 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ 分别是算子 $\partial_{xx} - 10x\partial_x$ 和 $\partial_{yy} - 100y\partial_y$的离散化。在使用指数 Runge–Kutta 格式进行时间积分时，我们便可以同时采用 PHIKS 和 PHISPLIT

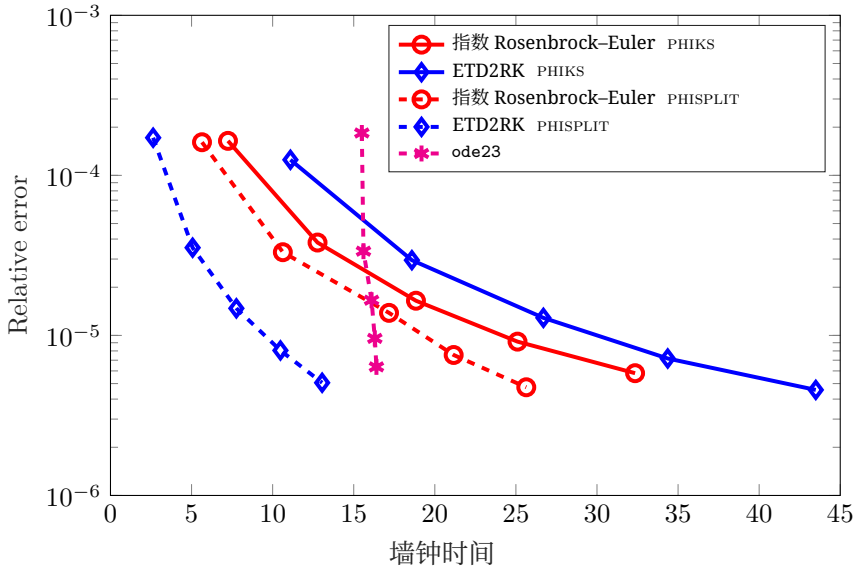


图 2：使用不同积分器和 $\hat{n} = 30$，将 Riccati 微分方程 (22) 求解至终止时间 $T = 0.025$ 时，在 Frobenius 范数下达到的误差以及以秒计的墙钟时间。各指数方法的时间步数见表 1。ode23 的输入容差（绝对容差和相对容差）分别为 5e−3、1e−3、1e−4、5.5e−5 和 5e−5。

指数 Rosenbrock–Euler PHIKS	步	15	30	45	60	75
	阶	–	2.10	2.06	2.04	2.03
ETD2RK PHIKS	步	10	20	30	40	50
	阶	–	2.12	2.07	2.05	2.04
指数 Rosenbrock–Euler PHISPLIT	步数	30	65	100	135	170
	阶	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	步数	30	65	100	135	170
	阶	–	2.05	2.03	2.02	2.02

表 2：使用不同的指数积分器和 $\hat{n} = 40$ 对 Riccati 微分方程 (22) 进行时间积分直至最终时间 $T = 0.025$ 时的时间步数量和观测收敛率。所达到的误差和墙钟时间显示于图 3 中。

这种方法使用尺寸更小的矩阵 D_1 和 D_2，随后在每个时间步利用四阶张量形式的 Tucker 算子构造近似。不过，由于这种方法仅适用于算子 (24) 的特定形式，因此本文不采用这种方法。

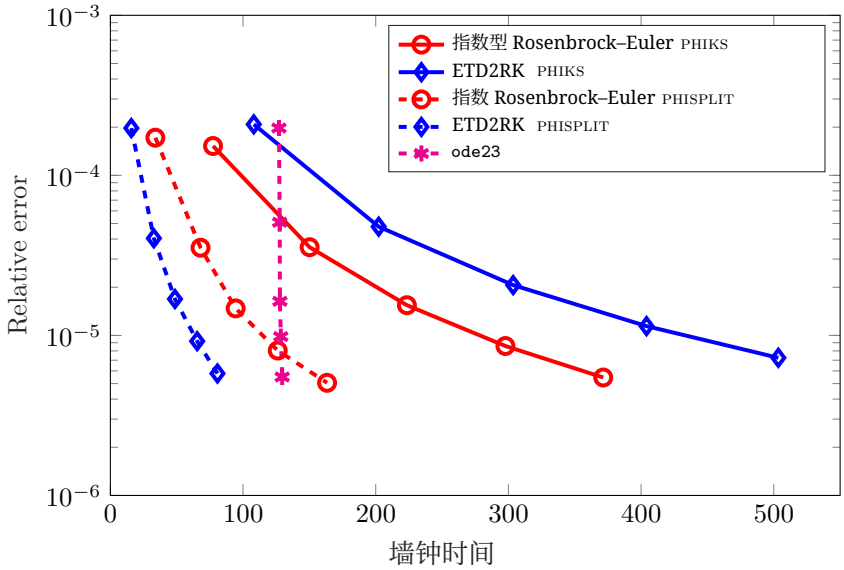


图 3：在不同积分器和 $n = 40$下，Riccati 微分方程 (22)的解截至最终时间 $T = 0.025$时所达到的 Frobenius范数误差和以秒为单位的墙钟时间。每种指数方法的时间步数量列于表2中。ode23的输入容差（绝对容差和相对容差）分别为 3e−3、4e−4、2.3e−4、9.5e−5 和 8.7e−5。

4.2 对流–扩散–反应

我们现在考虑以下三维演化型对流–扩散–反应（ADR）方程在空间上的半离散化（见参考文献 [8]）

$$\begin{cases} \partial_t u(t,x_1,x_2,x_3) = \varepsilon\Delta u(t,x_1,x_2,x_3) + \alpha(\partial_{x_1}+\partial_{x_2}+\partial_{x_3})u(t,x_1,x_2,x_3) \\ \qquad\qquad + g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)), \\ u_0(x_1,x_2,x_3) = 64x_1(1-x_1)x_2(1-x_2)x_3(1-x_3). \end{cases} \tag{25}$$

非线性函数 g 由下式给出

$$g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)) = \frac{1}{1+u(t,x_1,x_2,x_3)^2} + \Psi(t,x_1,x_2,x_3),$$

其中 $\Psi(t,x_1,x_2,x_3)$ 使得该方程的解析解为

$$u(t,x_1,x_2,x_3) = \mathrm{e}^t u_0(x_1,x_2,x_3).$$

该问题在区域 $[0,1]^3$ 内求解至终止时刻 $T = 1$，并配以齐次狄利克雷边界条件。其余参数设为 $\varepsilon = 0.75$ 和 $\alpha = 0.1$。通过采用二阶中心有限差分进行空间半离散，我们得到(1)型系统，其中 K 具有三维Kronecker和结构，而 A_μ 逼近 $\varepsilon\partial_{x_\mu x_\mu} + \alpha\partial_{x_\mu}$。我们首先进行数值模拟，分别对 x_1、x_2 和 x_3 变量采用 $n_1 = 40$、$n_2 = 41$和 $n_3 = 42$ 个内部离散点。时间积分采用四种方法：Lawson–Euler 格式(9)、指数 Euler 方法(11)、Lawson2b格式(10)和ETD2RK 方法(12)（见第3.2节中其实际

实现以及 PHISPLIT 版本）。特别地，对于Lawson格式，所需的矩阵指数exp(τA_μ)（ $\mu = 1$、2、3）在开始时一次性计算完成。随后，在时间积分过程中构造近似时，一阶格式和二阶格式分别需要每个时间步一个和两个Tucker算子。关于指数欧拉法和ETD2RK的 PHISPLIT 实现，我们同样在开始时间积分之前一次性计算所需的矩阵函数 $\varphi_1(\tau A_\mu)$和 $\varphi_2(\tau A_\mu)$，然后在每个时间步将它们适当地组合。与上述Lawson格式一样，该运算对一阶格式需要一个Tucker算子，对二阶格式需要两个，外加作用 $K\boldsymbol{u}_n$。

表3报告了两种方法在 PHISPLIT 和 PHIKS 实现下各自所需的时间步数量，而达到的相对误差和墙钟时间则汇总于图4中。首先，我们注意到，所有方法都呈现出预期的收敛阶，

Lawson–Euler	步骤	800	8800	16800	24800	32800
	阶	–	1.00	1.00	1.00	1.00
指数欧拉法 PHIKS	步骤	50	450	850	1250	1650
	阶	–	1.03	1.00	1.00	1.00
指数欧拉法 PHISPLIT	步	50	450	850	1250	1650
	阶	–	1.03	1.01	1.00	1.00
Lawson2b	步	1500	5500	9500	13500	17500
	阶	–	1.96	1.99	2.00	2.00
ETD2RK PHIKS	步骤	20	80	140	200	260
	阶	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	步骤	40	140	240	340	440
	阶	–	2.10	2.04	2.03	2.02

表3：使用不同指数积分器对ADR方程(25)的半离散化进行时间积分直至最终时间 $T = 1$ 时的时间步数及观测到的收敛率。此处我们分别对 x_1、 x_2 和 x_3 变量考虑了 $n_1 = 40$、 $n_2 = 41$ 和 $n_3 = 42$ 个内部空间离散点。所得误差与墙钟时间显示于图4。

结果也列于表3中。Lawson–Euler方法和指数Euler格式的PHIKS 实现（见图4上图）表现同样出色，尽管前者需要更多时间步。总体而言，在本实验中，指数 Euler 方法的 PHISPLIT 变体达到最高精度所需的时间大约快10倍。若考察二阶方法（图4下图），可以发现，Lawson2b格式达到与其他方法相同的精度需要更多墙钟时间；总体上，性能最佳的方法是 PHISPLIT 变体中的ETD2RK。图中还报告了使用内部 MATLAB ode23t积分器所获得的结果。它是一种三阶隐式龙格-库塔方法，采用变步长，适用于低精度要求下的刚性问题。尽管如此，它的表现仍不如所考察的指数积分器。

最后，我们针对 x_1、 x_2 和 x_3 变量的 $n_1 = 80$、 $n_2 = 81$ 和 $n_3 = 82$ 个内部离散点重复了实验。表4列出了每种方法所用的时间步数，而图5汇总了相对误差和墙钟时间。同样，我们注意到所有方法都呈现出预期的收敛阶，如表4所示，

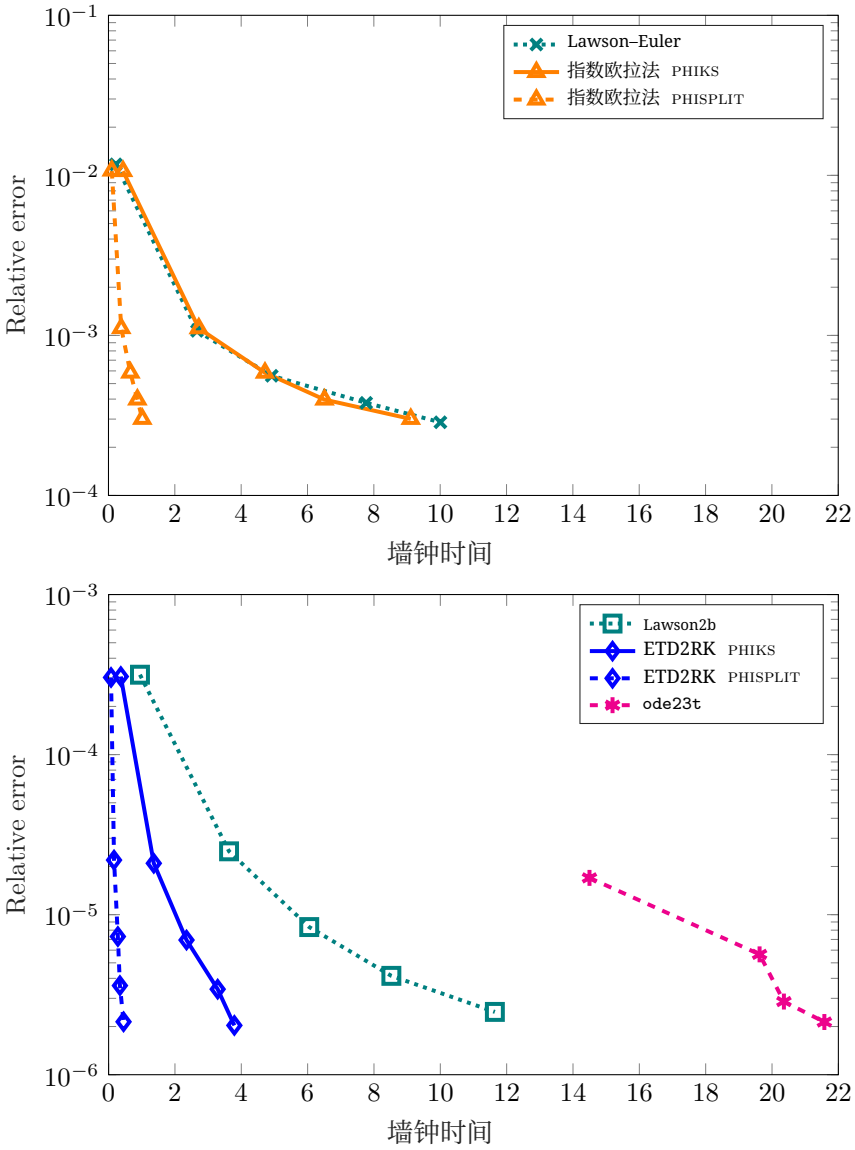


图4：在最终时间 $T = 1$ 之前，采用不同的一阶（上图）和二阶（下图）指数积分器求解ADR方程(25) 的半离散化时，在无穷范数下得到的误差和以秒为单位的墙钟时间。这里，对于 x_1、x_2 和 x_3 变量，我们分别考虑了 $n_1 = 40$、$n_2 = 41$和 $n_3 = 42$ 个内部空间离散点。每种指数方法的时间步数列于表3中。ode23t的输入容差（绝对容差和相对容差）分别为8e−3、4e−5、1e−5 和 5e−6。

尤其是在时间步长较大时，Lawson2b 方法会出现阶数降低。这是预料之中的，因为此时问题的刚性更强，而仅采用指数函数的格式会受到这一现象的影响（例如，参见参考文献 [17]）。随后，我们从图5中观察到，无论如何， PHISPLIT 方法都是

Lawson–Euler	步骤	800	8800	16800	24800	32800
	阶	–	1.00	1.00	1.00	1.00
指数欧拉法 PHIKS	步骤	50	450	850	1250	1650
	阶	–	1.02	1.00	1.00	1.00
指数欧拉法 PHISPLIT	步骤	50	450	850	1250	1650
	阶	–	1.03	1.01	1.00	1.00
Lawson2b	步骤	3000	4500	6000	7500	9000
	阶	–	1.79	1.87	1.92	1.94
ETD2RK PHIKS	步骤	20	80	140	200	260
	阶	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	步骤	40	140	240	340	440
	阶	–	2.10	2.04	2.03	2.02

表4：采用不同指数积分器对ADR方程(25)的半离散化进行时间积分直至最终时间 $T = 1$ 时的时间步数与观测收敛率。此处我们分别对 x_1、 x_2 和 x_3 变量考虑了 $n_1 = 80$、 $n_2 = 81$ 和 $n_3 = 82$ 个内部空间离散点。所达到的误差与墙钟时间见图5。

在所考虑的所有方法与技术中性能最佳，且在更严格的精度要求下加速比进一步增大。更具体地说，与其 PHIKS 对应实现相比， PHISPLIT实现大约快3.5倍，尽管（一般而言）它们需要更多时间步才能达到相当的精度。最后，Lawson格式始终性能最差。这主要是因为要达到其他方法的精度需要大量时间步，这一点在二阶格式中尤为明显（见表4底部和图5）。此外，ETD2RK 的 PHISPLIT 变体在不到10秒内达到了最严格的精度，而Lawson2b在100秒内仍无法达到误差大10倍的精度。因此，我们决定停止使用该积分器以更多时间步进行的数值模拟。最后，对于内部 MATLAB 常微分方程求解器套件，没有任何方法能在10分钟内输出解。

5 结论

本文介绍了一种基于 μ 模式的方法，可高效近似具有 d 维克罗内克和结构的矩阵上 φ 函数的作用。该技术称为 PHISPLIT，适用于使用指数积分器对初值常微分方程进行至多二阶的积分。它基于对时间推进格式中涉及的矩阵函数进行非精确方向分裂，同时保持方法的阶数。通过一个线性二次控制中的二维问题，以及一个三维对流–扩散–反应方程，使用多种指数积分器，展示了该方法相较于另一种以克罗内克形式计算 φ 函数作用的技术的有效性和优越性。未来值得探索的方向包括将该方法推广至更高阶积分器，并使用 PHISPLIT 技术进行图形处理器模拟；也可尝试单精度和/或半精度计算（这与时间积分误差的

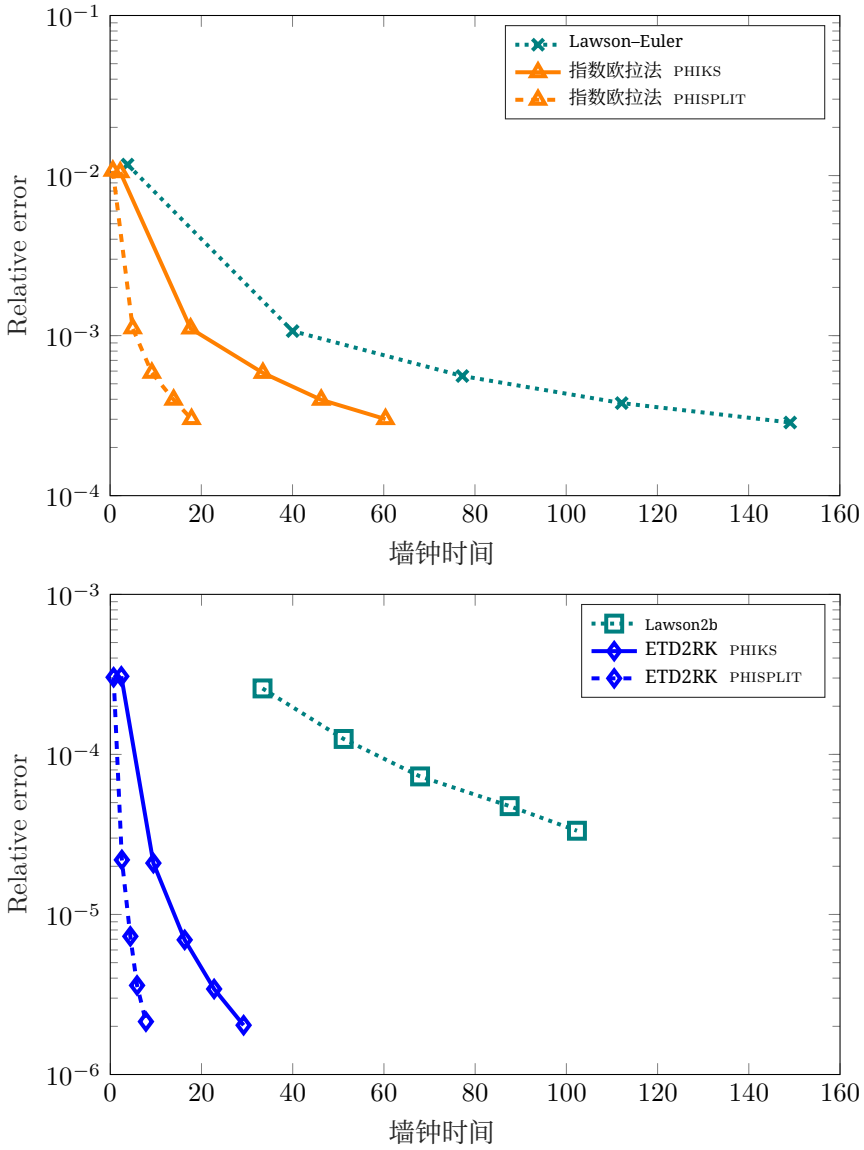


图5：对于 ADR 方程 (25) 的半离散化，在终止时间 $T = 1$ 前，采用不同的一阶（上）和二阶（下）指数积分器求解所得的无穷范数误差及墙钟时间（秒）。这里，$n_1 = 80$、$n_2 = 81$ 和 $n_3 = 82$ 分别表示 x_1、x_2 和 x_3 变量的内部空间离散点数。各方法所用的时间步数列于表 4。

大小相匹配），以处理科学与工程领域的不同问题。

致谢

作者感谢维罗纳大学基础研究计划 2019年项目（编号：RBVR199YFL）提供的部分资助，该项目题为“多智能体系统的几何演化”。

参考文献

[1] H. Abou-Kandil、G. Freiling、V. Ionescu 和 G. Jank。《控制与系统理论中的矩阵黎卡提方程》。Birkh¨auser Springer，巴塞尔，2003年。[2] A. H. Al-Mohy 和 N. J. Higham。《一种新的矩阵指数缩放平方算法》。SIAM J. Matrix Anal. Appl.，31(3):970–989，2010年。[3] A. H. Al-Mohy 和 N. J. Higham。《计算矩阵指数的作用及其在指数积分器中的应用》。SIAM J. Sci. Comput.，33(2):488–511，2011年。[4] H. Berland、B. Skaflestad 和 W. M. Wright。EXPINT——用于指数积分器的 MATLAB 软件包。技术报告 4，挪威科技大学，2005年。[5] H. Berland、B. Skaflestad 和 W. M. Wright。EXPINT——用于指数积分器的 MATLAB 软件包。ACM Trans. Math. Softw.，33(1):第 4 号文章，2007年。[6] M. Caliari、F. Cassini、L. Einkemmer、A. Ostermann 和 F. Zivcovich。一种用于求解克罗内克形式演化方程的 μ 模积分器。J. Comput. Phys.，455:110989，2022年。[7] M. Caliari、F. Cassini 和 F. Zivcovich。数值域狭窄的矩阵的矩阵指数近似。BIT Numer. Math.，60(4):1113–1131，2020年。[8] M. Caliari、F. Cassini 和 F. Zivcovich。一种用于指数积分器的 μ 模方法：克罗内克和的 φ 函数作用。arXiv 预印本 arXiv:2210.07667，2022年。[9] M. Caliari、F. Cassini 和 F. Zivcovich。BAMPHI：指数积分器中 φ 函数组合的无矩阵及无转置作用。J. Comput. Appl. Math.，423:114973，2023年。[10] M. Caliari、F. Cassini 和 F. Zivcovich。一种适用于多维张量结构问题的 μ 模 BLAS 方法。Numer. Algorithms，92(4):2483–2508，2023年。[11] M. Caliari、P. Kandolf、A. Ostermann 和 S. Rainer。重新审视 Leja 方法：矩阵指数的后向误差分析。SIAM J. Sci. Comput.，38(3):A1639–A1661，2016年。[12] M. Caliari 和 F. Zivcovich。泰勒算法近似矩阵指数的实时后向误差估计。J. Comput. Appl. Math.，346:532–548，2019年。[13] M. Croci 和 J. Mu˜noz-Matute。利用指数积分器中的克罗内克结构：通过求积快速近似矩阵的 φ 函数作用。J. Comput. Sci.，67:101966，2023年。[14] L. Einkemmer 和 A. Ostermann。克服扩散-反应分裂中的阶数降低。第一部分：狄利克雷边界条件。SIAM J. Sci. Comput.，37(3):A1577–A1592，2015年。

[15] S. Gaudreault、G. Rainwater 和 M. Tokman。KIOPS：一种用于指数积分器的快速自适应 Krylov 子空间求解器。J. Comput. Phys.，372:236–255，2018年。[16] C. Gonz´alez、A. Ostermann 和 M. Thalhammer。一种用于非自治抛物型问题的二阶 Magnus 型积分器。J. Comput. Appl. Math.，189(1–2):142–156，2006年。[17] M. Hochbruck、J. Leibold 和 A. Ostermann。关于 Lawson 方法在半线性刚性问题中的收敛性。Numer. Math.，145:553–580，2020年。[18] M. Hochbruck 和 A. Ostermann。指数积分器。Acta Numer.，19:209–286，2010年。[19] D. Li、S. Yang 和 J. Lan。高效且准确地计算指数积分器中出现的 φ 函数。Calcolo，59:第 11 号文章，2022年。[20] D. Li、X. Zhang 和 R. Liu。用于大规模刚性 Riccati 微分方程的指数积分器。J. Comput. Appl. Math.，389:113360，2021年。[21] D. Li、Y. Zhang 和 X. Zhang。计算 Lyapunov 算子的 φ 函数及其在矩阵值指数积分器中的应用。Appl. Numer. Math.，182:330–343，2022年。[22] V. T. Luan、J. A. Pudykiewicz 和 D. R. Reynolds。进一步开发适用于气象模型的高效准确时间积分格式。J. Comput. Phys.，376:817–837，2019年。[23] H. Mena、A. Ostermann、L.-M. Pfurtscheller 和 C. Piazzola。矩阵微分方程的数值低秩近似。J. Comput. Appl. Math.，340:602–614，2018年。[24] J. Mu˜noz-Matute、D. Pardo 和 V. M. Calo。利用指数积分器中 φ 函数的 Kronecker 积结构。Int. J. Numer. Methods Eng.，123(9):2142–2161，2022年。[25] J. Niesen 和 W. M. Wright。算法 919：一种用于计算指数积分器中出现的 ϕ 函数的 Krylov 子空间算法。ACM Trans. Math. Softw.，38(3):1–19，2012年。[26] T. Penzl。LYAPACK：用于大型 Lyapunov 方程和 Riccati 方程、模型降阶问题以及线性二次最优控制问题的 MATLAB 工具箱。用户指南（1.0 版），2000年。[27] F. Rousset 和 K. Schratz。低正则性积分器的通用框架。SIAM J. Numer. Anal.，59(3):1735–1768，2021年。[28] J. Sastre、J. Ib´a˜nez 和 E. Defez。加速矩阵指数的计算。Appl. Math. Comput.，340:206–220，2019年。[29] B. Skaflestad 和 W. M. Wright。与指数相关的矩阵函数的缩放与修正平方方法。Appl. Numer. Math.，59(3–4):783–799，2009年。[30] C. F. Van Loan。无处不在的 Kronecker 积。J. Comput. Appl. Math.，123(1–2):85–100，2000年。