SIAM J. SCI. COMPUT. © 2019 Society for Industrial and Applied Mathematics
Vol. 41, No. 5, pp. A2999–A3020

PARALLEL-IN-TIME MAGNUS INTEGRATORS[*]

B. T. KRULL[†] AND M. L. MINION[‡]

Abstract. Magnus integrators are a subset of geometric integration methods for the numerical solution of ordinary differential equations that conserve certain invariants in the numerical solution. This paper explores temporal parallelism of Magnus integrators, particularly in the context of nonlinear problems. The approach combines the concurrent computation of matrix commutators and exponentials within a time step with a pipelined iteration applied to multiple time steps in parallel. The accuracy and efficiency of time-parallel Magnus methods up to order six are highlighted through numerical examples and demonstrate that significant parallel speedup is possible compared to serial methods.

Key words. ordinary differential equations, Magnus expansion, isospectral flow, nonlinear ordinary differential equations

AMS subject classifications. 34L30, 65L05, 65Y05

DOI. 10.1137/18M1174854

1. Introduction.

The solution of ordinary differential equations (ODEs) is a well established field with applications across the spectrum of scientific disciplines. Numerical methods date back at least to Euler's work in 1768 [5], and the accuracy, stability, and efficiency of various methods is well studied (see, for example, [8, p. 447][9]). In more recent years, the study of specialized numerical methods for ODEs that preserve certain mathematical properties of the numerical solution has seen increased interest. Examples include methods for Hamiltonian systems that numerically conserve invariants of the true dynamical system such as the energy or angular momentum. More generally, the true solution of an ODE posed in N-dimensional space may reside for all time on a manifold $\mathcal{M}$ of dimension $d < N$, and the goal is to devise a method for which the numerical solution will also remain on $\mathcal{M}$. Such methods are referred to in general as *geometric integrators*. The interested reader is encouraged to consult [7] for a comprehensive introduction to the subject.

As a concrete example, consider the ODE given by

$$\frac{d}{dt}Y(t) = F(Y(t)), \qquad Y(0) = Y_0, \tag{1.1}$$

where Y and $F(Y)$ are both $N \times N$ matrices. Depending on the form of F, certain properties of the initial value Y_0 may be preserved for all time, such as the determinant, orthogonality, idempotency, or the spectrum of eigenvalues. In general, standard numerical methods such as linear multistep or Runge–Kutta methods will not produce solutions that conserve such properties [16].

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section March 12, 2018; accepted for publication (in revised form) August 15, 2019; published electronically October 1, 2019.
https://doi.org/10.1137/18M1174854
Funding: This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contract DE-AC02005CH11231. Part of the simulations were performed using resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

[†]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (bkrull@lbl.gov).

[‡]Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (mlminion@lbl.gov).

Magnus integrators are a subset of geometric integrators based on the expansion proposed by Wilhelm Magnus in 1954 [14]. The Magnus expansion is closely tied to the concept of Lie groups and algebras due to the presence of matrix commutators, also known as Lie brackets. The discussion concerning solutions to (1.1) can in fact be generalized to differential equations on Lie groups (see, e.g., [12]); however, in this work we strictly use matrix valued solutions. Magnus integrators can also be viewed as a type of exponential integrator (see, e.g., [11] for a review) since they require that the matrix exponential be evaluated. The comprehensive review of the Magnus expansion and applications by Blanes et al. [1] provides a description of several physical applications to which the Magnus expansion has been applied, including nuclear, atomic, and molecular dynamics; nuclear magnetic resonance; quantum field theory and high energy physics; electromagnetism; optics; geometric control of mechanical systems; and the search for periodic orbits.

In general, numerical Magnus integrators are constructed by applying quadrature rules of suitable order to a truncation of the Magnus expansion. The review [12] presents various types of Magnus integrators up to sixth-order, and methods of order up to eight are considered in [2]. As the order increases, the number of commutator terms required in the Magnus expansion grows quickly; hence in these papers and others, a detailed discussion of how to minimize the number of commutators required for a given order is presented. So-called commutator-free Magnus integrators (e.g., [3, 18]) have also been proposed that replace the need to compute matrix commutators with additional matrix exponentials. This can reduce the total computational cost of the method depending on the relative cost of computing commutators versus matrix exponentials. In this paper, an additional avenue for reducing the time to solution for Magnus integrators is investigated, namely parallelization in time.

The study of parallel numerical methods for ODEs dates back at least to Nievergelt in 1964 [17], and the field has seen an increase in activity in the last 15 years. (See [6] for a recent review.) The standard classification of parallel methods for ODEs includes parallelism across the method, across the problem, and across the time steps [4]. In this work we demonstrate the utility of parallelization across both the method and the time steps for Magnus integrators for both linear and nonlinear equations. Special attention is paid to schemes for solving nonlinear differential equations for isospectral flow problems, although the methodology that is described can be applied more generally.

The remainder of this paper is organized as follows. Section 2 presents the mathematical background behind the Magnus expansion and Magnus integrators followed by some specific Magnus integrators based on Gaussian collocation in section 3. The parallelization strategies for these integrators is presented in section 4. Numerical results comparing the efficiency of different parallel methods is presented in section 5. The results demonstrate that significant parallel speedup over serial methods is possible and show that parallel higher-order methods are superior in terms of accuracy and time to solution compared to lower-order methods.

2. Mathematical preliminaries.

In this section, a review of the mathematics behind the construction of Magnus integrators for both linear and nonlinear problems is reviewed.

2.1. Matrix calculus and differential equations.

Given a constant matrix $A \in \mathbb{F}^{N\times N}$, where $\mathbb{F}$ can be either $\mathbb{R}$ or $\mathbb{C}$, the exponential of A is defined by the power series

$$e^A = \sum_{n=0}^{\infty} \frac{A^n}{n!}. \tag{2.1}$$

Given the time-dependent vector $y(t) \in \mathbb{F}^N$, the solution to the differential equation

$$y'(t) = Ay(t), \qquad y(0) = y_0 \tag{2.2}$$

is given by

$$y(t) = e^{At} y_0. \tag{2.3}$$

This is easily shown by differentiating the power series definition (2.1) of the exponential on the right-hand side term-by-term to give

$$\frac{d}{dt} e^{At} = Ae^{At}. \tag{2.4}$$

Now consider the more general case of a time-dependent matrix $A(t)$. Again differentiating definition (2.1) term-by-term and using the product rule yields

$$\frac{d}{dt} e^{A(t)} = A'(t) + \frac{A(t)A'(t) + A'(t)A(t)}{2!} + \frac{A(t)'A(t)^2 + A(t)A'(t)A(t) + A(t)^2 A'(t)}{3!} \cdots. \tag{2.5}$$

The right-hand side of (2.5) can be rearranged to give

$$\frac{d}{dt} e^{A(t)} = \operatorname{dexp}_{A(t)}(A'(t)) e^{A(t)}, \tag{2.6}$$

where the operator on the right is defined by

$$\operatorname{dexp}_{A(t)}(X(t)) = X(t) + \frac{[A(t), X(t)]}{2!} + \frac{[A(t), [A(t), X(t)]]}{3!} \cdots, \tag{2.7}$$

and brackets $[\cdot, \cdot]$ denote the matrix commutator $[A, X] = AX - XA$.

Now consider the linear system of ODEs

$$y'(t) = A(t)y(t), \qquad y(0) = y_0. \tag{2.8}$$

To find a solution, suppose that it can be written in the form

$$y(t) = e^{\Omega(t)} y_0 \tag{2.9}$$

for some matrix $\Omega(t)$. Using (2.6), $\Omega(t)$ satisfies

$$\operatorname{dexp}_{\Omega(t)}(\Omega'(t)) = A(t), \qquad \Omega(0) = 0. \tag{2.10}$$

The same formal derivation can be applied to a nonlinear system of equations

$$y'(t) = A(y(t), t)y(t), \qquad y(0) = y_0. \tag{2.11}$$

Again, if the solution to this equation is to take the form of (2.9), then $\Omega(t)$ satisfies

$$\operatorname{dexp}_{\Omega(t)}(\Omega'(t)) = A(y(t), t), \qquad \Omega(0) = 0. \tag{2.12}$$

In the next section, methods for finding $\Omega(t)$ are considered.

2.2. Magnus expansion. In 1954, Magnus introduced an explicit expression for the solution of (2.8), which is reviewed here [14]. The first step is the inversion of the operator $\mathrm{dexp}_{\Omega(t)}$, which gives a differential equation for $\Omega(t)$

$$\Omega'(t) = \sum_{n=0}^{\infty} \frac{B_n}{n!} \mathrm{ad}^n_{\Omega(t)}(A(t)), \qquad \Omega(0) = 0, \tag{2.13}$$

where the B_n are the Bernoulli numbers and

$$\mathrm{ad}^k_{\Omega}(X) = [\Omega, \mathrm{ad}^{k-1}_{\Omega} X], \quad \mathrm{ad}^0_{\Omega}(X) = X. \tag{2.14}$$

Next, a Picard-type iteration is applied to (2.13),

$$\Omega^{k+1}(t) = \int_0^t \sum_{n=0}^{\infty} \frac{B_n}{n!} \mathrm{ad}^n_{\Omega^k(t)}(A(t)). \tag{2.15}$$

Collecting terms in (2.15) yields an infinite series for $\Omega(t)$

$$\Omega(t) = \Omega^{(1)}(t) + \Omega^{(2)}(t) + \Omega^{(3)}(t) + \dots, \tag{2.16}$$

where

$$\Omega^{(1)}(t) = \int_0^t d\tau A(\tau), \tag{2.17}$$

$$\Omega^{(2)}(t) = \frac{1}{2} \int_0^t d\tau_2 \int_0^{\tau_2} d\tau_1 [A(\tau_2), A(\tau_1)], \tag{2.18}$$

$$\Omega^{(3)}(t) = \frac{1}{6} \int_0^t d\tau_3 \int_0^{\tau_3} d\tau_2 \int_0^{\tau_2} d\tau_1 [A(\tau_3), [A(\tau_2), A(\tau_1)]] + [[A(\tau_3), A(\tau_2)], A(\tau_1)], \tag{2.19}$$

$$\begin{aligned} \Omega^{(4)}(t) = \frac{1}{12} \int_0^t d\tau_4 \int_0^{\tau_4} d\tau_3 \int_0^{\tau_3} d\tau_2 \int_0^{\tau_2} d\tau_1 & \left[[[A(\tau_4), A(\tau_3)], A(\tau_2)\right], A(\tau_1)] \\ & + [A(\tau_4), [[A(\tau_3), A(\tau_2)], A(\tau_1)]] \\ & + [A(\tau_4), [A(\tau_3), [A(\tau_2), A(\tau_1)]]] \\ & + [A(\tau_3), [A(\tau_2), [A(\tau_1), A(\tau_4)]]] . \end{aligned} \tag{2.20}$$

Each subsequent term in the series contains an additional integration operator, commutators of one higher order, as well as an increasing number of commutator terms. The reader is referred to the original work of Magnus [14] or the extensive review [1] for further details. To summarize, the Magnus expansion gives an explicit formula for the solution of the linear equation given by (2.8) in the form of the exponential of a matrix defined by an infinite series given by (2.16).

2.3. The magnus expansion for nonlinear problems. The same formal procedure used in the last section to construct the solution to the linear problem (2.8) can also be applied to the nonlinear system (2.11). One can still represent the solution in terms of the exponential of the function $\Omega(t)$, and the only difference is that in the Magnus expansion terms given above in (2.17)–(2.20), the terms $A(\tau)$ must be replaced with $A(y(\tau), \tau)$, which by the definition of the solution is $A(e^{\Omega(\tau)}y_0, \tau)$. Although this may appear at first a small change in notation, the implication is quite important. For the nonlinear problem, the Magnus expansion does not give an explicit formula for the function $\Omega(t)$ as in the linear case. Instead, the result is an equation for $\Omega(t)$ involving an infinite expansion of terms containing integrals of commutators dependent on $\Omega(t)$. The central insight of this paper is that this equation for $\Omega(t)$ can be solved efficiently by a fixed point iteration that is readily amenable to parallelization in the time direction.

2.4. Isospectral flows. A special type of matrix differential equation for which the eigenvalues of the solution are independent of time is called *isospectral flow*. Problems of this form exist in application domains including electronic structure, wave dynamics, and linear algebra. Isospectral flow is often associated with the concept of a Lax pair: two matrices or operators $A(t), Y(t)$ dependent on time that satisfy the Lax equation

$$Y'(t) = [A(Y,t), Y(t)], \quad Y(0) = Y_0, \tag{2.21}$$

where $Y(t), A(Y,t) \in \mathbb{F}^{N\times N}$.

It is straightforward to show that the solution to (2.21) can be written in the form of the transformation

$$Y(t) = (e^{\Omega(t)})Y_0(e^{\Omega(t)})^{-1}, \tag{2.22}$$

where $\Omega(t)$ is defined by the Magnus expansion with respect to $A(t)$. Since the form of $Y(t)$ takes on a similarity transformation, the eigenvalues of $Y(t)$ do not change in time, hence the term *isospectral*. In the special case where A is Hermitian (or self-adjoint) and Y is skew-Hermitian (or skew-adjoint), the exponential $e^{\Omega(t)}$ is unitary, which reduces (2.22) to

$$Y(t) = (e^{\Omega(t)})Y_0(e^{\Omega(t)})^{\dagger}. \tag{2.23}$$

3. Numerical methods based on the Magnus expansion.

In this section, the process for constructing numerical methods for differential equations based on the Magnus expansion is discussed. In general, numerical methods are constructed by designing appropriate quadrature rules for the Magnus expansion truncated to a given order. The presentation here is focused on collocation type schemes based on Gaussian quadrature rules. As proved in [13], quadrature rules for the terms in the Magnus expansion based on s Gauss–Legendre quadrature nodes are sufficient for constructing a method of order $2s$. Here, methods of orders two, four, and six are considered using both Gauss–Legendre and Gauss–Lobatto quadrature nodes. These methods correspond to quadrature rules applied to one, two, and four terms, respectively, in (2.16).

Considerable attention in the literature on Magnus integrators is devoted to designing methods requiring the minimum number of function evaluations and matrix commutators for a given order of accuracy [13, 1, 12]. In the context of time parallelization, the manner in which the cost of commutators and function evaluations are counted must reflect the fact that much of the work can be done in parallel, and the minimum parallel cost is not necessarily achieved by a direct parallelization of the serial method with the fewest commutators.

Methods for linear equations are discussed first, followed by a discussion of additional considerations for nonlinear problems in section 3.3.

TABLE 1
Quadrature nodes and weights for Gauss–Legendre and Gauss–Lobatto rules.

Name	Order	Nodes	$q^{(1)}$
Lob-2	2	$0,\ 1$	$\frac{1}{2},\ \frac{1}{2}$
Lob-3	4	$0,\ \frac{1}{2},\ 1$	$\frac{1}{6},\ \frac{4}{6},\ \frac{1}{6}$
Leg-3	6	$\frac{1}{2}-\frac{1}{2}\sqrt{\frac{3}{5}},\ \frac{1}{2},\ \frac{1}{2}+\frac{1}{2}\sqrt{\frac{3}{5}}$	$\frac{5}{18},\ \frac{8}{18},\ \frac{5}{18}$

3.1. Quadrature rules for the Magnus expansion. In this section, the specific types of quadrature rules used in the numerical methods are described. Quadrature rules based on Gauss–Lobatto or Gauss–Legendre quadrature rules using either two or three quadrature nodes are considered here. Table 1 lists the specific nodes used for each choice as well as the accompanying classical weights. For a method of a given order, each term in the truncated Magnus expansion must be approximated using the function values $A_m = A(t_m)$ (or $A_m = A(y(t_m), t_m)$ for nonlinear problems) at the quadrature nodes t_m corresponding to the quadrature nodes scaled to the time step interval $[t_n, t_{n+1}]$. For the schemes described below, the same quadrature nodes are used at each term in the expansion in (2.16).

First consider the approximation to the first term of the expansion $\Omega^{(1)}(t)$ on the interval $[t_n, t_{n+1}]$ with $\Delta t = t_{n+1} - t_n$. Approximating the integral by Gaussian quadrature gives

$$\Omega^{(1)}(t_{n+1}) = \int_{t_n}^{t_{n+1}} A(t)\, dt \approx \Delta t \sum_{j=1}^{M} q_j^{(1)} A_j = \Omega_{n+1}^{(1)}, \tag{3.1}$$

where M is the number of quadrature nodes. This is classical quadrature, and the well-known coefficients $q_j^{(1)}$ are given for completeness in Table 1.

In order to obtain a fourth-order method, the second term in the Magnus expansion must be included. The simplest approximation to $\Omega_{n+1}^{(2)}$ sufficient for fourth-order accuracy requires the calculation of only a single commutator term

$$\Omega_{n+1}^{(2)-1} = \Delta t^2 q^{(2)-1}[A_1, A_3], \tag{3.2}$$

with $q^{(2)-1} = 1/12$. The method denoted Lob-4-1 (where the 1 denotes one commutator term) uses this approximation. To compute $\Omega^{(2)}$ to the accuracy required for a sixth-order method, three nodes can be used and it is necessary to compute three commutators

$$\Omega_{n+1}^{(2)-3} = q_1^{(2)-3}[A_1, A_2] + q_2^{(2)-3}[A_1, A_3] + q_3^{(2)-3}[A_2, A_3] \tag{3.3}$$

with the values

$$q_j^{(2)-3} = [-7.1721913818656\text{e}{-2}, -3.5860956909328\text{e}{-2}, -7.1721913818656\text{e}{-2}]\,. \tag{3.4}$$

Despite the increased computational cost of two additional commutators, in a parallel implementation all three commutators can be computed simultaneously.

To achieve sixth-order accuracy, the first four terms of the Magnus expansion must be included. The sixth-order method denoted Leg-6 approximates the $\Omega^{(3)}$ term using three Gauss–Legendre nodes following the discussion in [12]. Specifically,

$$\begin{aligned}\Omega^{(3)}_{n+1} = \Delta t^3 \Big(&\left[q^{(3)}_{1,1}A_1 + q^{(3)}_{1,2}A_2 + q^{(3)}_{1,3}A_3, [A_1, A_2]\right] \\ &+ \left[q^{(3)}_{2,1}A_1 + q^{(3)}_{2,2}A_2 + q^{(3)}_{2,3}A_3, [A_1, A_3]\right] \\ &+ \left[q^{(3)}_{3,1}A_1 + q^{(3)}_{3,2}A_2 + q^{(3)}_{3,3}A_3, [A_2, A_3]\right]\Big). \qquad (3.5)\end{aligned}$$

The values of the coefficients $q^{(3)}_{i,j}$ are the same as those in [12] and in matrix form are
(3.6)

$$q^{(3)} = \begin{bmatrix} 3.4538506760729\text{e}{-3} & -5.5849500293944\text{e}{-3} & -7.1281599059377\text{e}{-3} \\ 1.6534391534391\text{e}{-3} & 0.0 & -1.6534391534391\text{e}{-3} \\ 7.1281599059377\text{e}{-3} & 5.5849500293945\text{e}{-3} & -3.4538506760729\text{e}{-3} \end{bmatrix}.$$

Exploiting the linear property of commutators ($[A, X] + [A, Y] = [A, X + Y]$) allows one to combine terms that share the same inner single commutator and reduce the number of commutators from nine to three. Note that the single commutator terms, i.e., $[A_1, A_2]$, are computed during the formation of the $\Omega^{(2)}_{n+1}$ term and need not be computed again.

The fourth term in the Magnus expansion can be approximated using a low-order quadrature for a sixth-order method. Following the discussion in [2], the fourth term is approximated by

$$\Omega^{(4)}_{n+1} = \Delta t^4 q^{(4)}[B_0, [B_0, [B_0, B_1]]], \qquad (3.7)$$

where $q^{(4)} = 1/60$ and

$$B_i = \Delta t \sum_{j=1}^{3} q^{(1)}_j (t_j - 0.5)^i A_j \qquad (3.8)$$

with $q^{(1)}_j$ given in the last row of Table 1.

In section 5, numerical examples are presented for five different Magnus integrators. Table 2 lists the specific discretization of each term included for a given method. The overall order of each method is determined by either the number of terms used in the expansion or the order of the quadrature rules. For example, the methods Leg-2, Leg-4-3, and Leg-6 use the same quadrature nodes but differ in the number of terms used in the expansion, while Lob-2 and Leg-2 use different nodes but are both second-order because only one term in the expansion is used.

3.2. The matrix exponential and solution update. Once all of the quadrature approximations are applied and the value of Ω_{n+1} is computed, the solution can be updated by

$$y_{n+1} = e^{\Omega_{n+1}} y_n. \qquad (3.9)$$

Table 2
Description of the numerical schemes.

Name	Order	Nodes	Ω
Lob-2	2	Lob-2	$\Omega^{(1)}$
Leg-2	2	Leg-3	$\Omega^{(1)}$
Lob-4-1	4	Lob-3	$\Omega^{(1)} + \Omega^{(2)-1}$
Leg-4-3	4	Leg-3	$\Omega^{(1)} + \Omega^{(2)-3}$
Leg-6	6	Leg-3	$\Omega^{(1)} + \Omega^{(2)-3} + \Omega^{(3)} + \Omega^{(4)}$

There are many approaches to computing the product of a matrix exponential and a vector of the form $e^A y$ [15], some of which explicitly compute the term e^A and some that only approximate the product $e^A y$. The choice of method is problem dependent and does not affect the discussion of time parallelism of the methods. In the numerical examples presented here, the scaling-and-squaring method from [10] is used to form the matrix exponential explicitly.

3.3. Considerations for nonlinear ODEs. Consider now problems of the form

$$y' = A(y,t)y, \tag{3.10}$$

where nonlinearity is introduced through the y-dependence of A. The terms in the Magnus expansion approximations introduced above now depend on the solution through A_m and cannot simply be evaluated. The numerical solution at each quadrature node t_m will be denoted y_m, and hence $A_m = A(y_m, t_m)$. The values of y_m at each quadrature node are computed by

$$y_m = e^{\Omega_m} y_n, \tag{3.11}$$

where Ω_m is an approximation to the Magnus expansion on the interval $[t_n, t_m]$. The construction of Ω_m is discussed below.

A simple fixed point Picard-type iteration is used to simultaneously solve for the values Ω_m and y_m. The iterative scheme is initialized by setting $y_m^{k=1} = y_n$ at each node m, where k denotes the iteration. The solution at each quadrature node is updated by

$$y_m^{k+1} = e^{\Omega_m^k} y_n. \tag{3.12}$$

Then Ω_m^{k+1} is computed using values $A(y_m^{k+1}, t_m)$ as described below.

To compute Ω_m^k, the process for constructing the quadrature rules in section 3.1 needs to be applied to each quadrature node. Evaluating the values $A(y_m^k, t_m)$ at each node t_m is straightforward but needs to be performed each iteration. In all cases considered here, the same matrix commutators are used for each quadrature rule, so computing the commutators is also identical to the linear case. The significant difference is that a quadrature rule for each term in the Magnus expansion must be computed for each interval $[t_n, t_m]$ rather then just $[t_n, t_{n+1}]$ as in the linear case. The coefficients for each of the terms are included in Appendix A.

Once each Ω_m^k is computed, the matrix exponential can be computed for each node, and a new solution is obtained at each node by (3.12). The solution is considered converged when the maximum absolute value of the residual

$$R_m^k = e^{\Omega_m^k} y_n - y_m^k = y_m^{k+1} - y_m^k \tag{3.13}$$

is less than a predefined tolerance. In a traditional implementation of this iteration using Gauss–Legendre nodes, it is not necessary to compute the values Ω_{n+1}^k and y_{n+1} at the end of the time step during the iterations; however, when pipelining of iterations is employed as discussed in the next section, y_{n+1} is computed each iteration to update the initial condition for the next time step.

3.4. Considerations for isospectral flows. Consider now problems of the form of (2.21). The procedure for constructing Magnus integrators follows exactly that laid out in the previous section except that the solution is defined at quadrature nodes by

$$Y_m^{k+1} = e^{\Omega_m^k} Y_n e^{-\Omega_m^k}, \tag{3.14}$$

and likewise for the computation of Y_{n+1}^{k+1} from Ω_{n+1}^k.

4. Parallelization in time for Magnus integrators. In this section, we investigate the theoretical computational cost of the Magnus integrators introduced in the previous section in both serial and parallel settings. We consider both parallelization across the method and parallelization across the time steps. In the following discussion, it is assumed that arbitrarily many processors are available for a given problem and the cost of communication between processors is ignored. It is also assumed that the matrix exponential is formed explicitly as is done for the numerical results given in section 5.

4.1. The linear case. First consider linear problems where $A(t)$ does not depend on the solution y. For each of the N time steps, the following tasks must be performed:

L1. Evaluate $A_m = A(t_m)$ for each quadrature node t_m.
L2. Compute commutators necessary for each term in the truncated Magnus expansion.
L3. Apply quadrature rules to compute $\Omega(t_{n+1})$.
L4. Form the exponential of $\Omega(t_{n+1})$.
L5. Compute the solution y_{n+1} from y_n by matrix multiplication.

Denote by C_A the computational cost of computing $A(t)$ for a given time. Then if M quadrature nodes are used, L1 has a computational cost of MC_A. Next let n^p denote the number of commutators required to compute the pth term in the Magnus expansion and C_C the cost of computing one commutator. Assuming that each commutator in the $p+1$ term can be formed with one additional commutator applied to a term from term p, the total number of commutators to compute is simply $n^1 + \dots n^P$. Denoting this sum by N_C, the cost of L2 is $N_C C_C$. Task L3 requires only that a linear combination of the terms computed in L2 be computed. We can denote this cost by $N_C C_L$, where C_L is the cost of adding a term in the linear combination. Denote by C_E the cost of the matrix exponential, and hence the cost of L4 is C_E. Likewise denote by C_M the cost of multiplying the solution by a matrix which corresponds to the cost of task L5. Putting these together, the serial cost for N time steps is

$$\mathbf{C_S} = N(MC_A + N_C(C_C + C_L) + C_E + C_M). \tag{4.1}$$

Now consider the parallelization of the method for the linear problem across the time steps. In task L1, each function evaluation can be done concurrently, so that the cost is reduced from MC_A to C_A. For task L2, all the commutators of a given order can be computed concurrently, so that cost is reduced from $N_C C_C$ to PC_C. Task L3 can be done with cost $\log_2(N_C)C_L$, and the cost of task L4 and L5 remains the same.

Next consider the cost when both parallelization across the method and across the time steps is employed. Given sufficient processors, tasks L1–L4 can all be computed on all time steps concurrently. Only task L5 must be done serially so that the total cost using both forms of parallelism becomes

$$\mathbf{C_P} = C_A + PC_C + \log_2(N_C)C_L + C_E + NC_M. \tag{4.2}$$

Clearly this is a significant reduction in computational cost. If the cost of computing the commutators and matrix exponential (L2 and L4) dominates the other terms, the theoretical parallel speedup approaches N.

An important point to make about this counting is that the cost per step when using parallelization across the method depends very little on the number of quadrature nodes or commutators used in each term since only the cost of task L3 depends on these factors. Hence higher-order methods are only modestly more expensive per step than lower-order methods and there is less benefit from reducing the number of commutators required in each term of the Magnus expansion since multiple terms can be computed in parallel. Furthermore, for a given accuracy, higher-order methods will typically require fewer time steps (i.e., smaller N).

4.2. The nonlinear case. For nonlinear problems, the theoretical accounting of cost must be modified somewhat. Since we are using an iterative procedure to compute $\Omega(t_m)$, the following steps must be done for each iteration in each time step:

N1. Evaluate $A(y_m^k, t_m)$ for each quadrature node t_m.
N2. Compute commutators necessary for each term in the truncated Magnus expansion.
N3. Apply quadrature rules to compute Ω_m^k at each quadrature node t_m.
N4. Form exponential of Ω_m^k at each quadrature node t_m.
N5. Compute y_m^{k+1} at each quadrature node by matrix multiplication by Ω_m^k.

The main difference between these tasks and the linear case is that N2–N5 are done for each quadrature node instead of only once. For simplicity, the serial cost of these steps will be assumed to be M times that of the linear case. Hence, denoting by K_S the number of iterations required for each step in a serial implementation, the serial cost for the nonlinear Magnus method iteration becomes

$$\mathbf{C_S} = NK_SM(C_A + N_C(C_C + C_L) + C_E + C_M). \tag{4.3}$$

As will be shown below, the number of iterations required for convergence K_S depends on Δt in a nontrivial way.

If we allow parallelization across the method, the tasks above can all be computed concurrently at each quadrature node, and hence the cost of each iteration for the nonlinear method is essentially that of one step in the linear case using parallelization across the method.

$$\mathbf{C_I} = C_A + PC_C + \log_2(N_C)C_L + C_E + C_M. \tag{4.4}$$

The speedup across the method is bounded by M.

Now consider parallelization across the time steps. The simplest way that this can be accomplished is to pipeline the iterations. We first divide the N time steps into blocks of size N_P with each step in a block assigned to a group of processors indexed by n_p. At each time step n_p for each iteration k, the initial condition is assigned the final value from iteration $k-1$ of the time step $n_p - 1$. For each block, $N_P - 1$ pipelined iterations are required before the last processor has a consistent initial condition. After this initialization step, assume K_P additional iterations are needed for convergence on every time step in the block. As in the serial case K_P depends in general on the time step Δt and now also on N_P, increasing as N_P increases and decreasing at Δt decreases. Furthermore $K_P \geq K_S$.

The parallel cost on each block will be $(N_P - 1 + K_P)\mathbf{C_I}$ compared to $(N_P K_S)\mathbf{C_I}$ when no parallelization across the time steps is applied (i.e., $N_P = 1$). The potential speedup from parallelization across the time steps is then

$$\mathbf{S} = \frac{N_P K_S}{N_P - 1 + K_P} = \frac{K_S}{1 + (K_P - 1)/N_P}. \tag{4.5}$$

Clearly the speedup is bounded by the number of serial iterations required and the best speedup will occur when the quantity K_P/N_P remains small as N_P increases. This ratio will be investigated in the numerical examples.

5. Numerical examples.

In this section, the performance of the different Magnus integrators introduced in section 3 is examined in both serial and parallel settings. First, the performance of the methods is considered as applied to a Toda lattice problem from the literature. Serial results demonstrating the relative accuracy and efficiency of the various methods are presented. In section 5.1.3, some preliminary results on parallelization of the methods are presented. In section 5.2, the parallel performance of the methods is considered on a problem motivated by real-time time-dependent density functional theory (RT-TDDFT).

5.1. Test Case 1: The periodic Toda lattice.

The numerical methods will first be evaluated on the test problem of a d-particle periodic Toda lattice [19], a one-dimensional chain whose dynamics are governed by nonlinear nearest-neighbor interactions. The equations of motion are a Hamiltonian system for positions q_j and momenta p_j (assuming unit masses)

$$q_j' = p_j, \tag{5.1}$$

$$p_j' = e^{-(q_j - q_{j-1})} - e^{-(q_{j+1} - q_j)}. \tag{5.2}$$

In order to cast the dynamics in terms of a Lax pair as in (2.21), one uses the Flaschka change of variables

$$\alpha_j = \frac{1}{2} e^{-(q_{j+1} - q_j)/2}, \tag{5.3}$$

$$\beta_j = \frac{1}{2} p_j, \tag{5.4}$$

which leads to definitions of Y and A

(5.5)

$$Y = \begin{bmatrix} \beta_1 & \alpha_1 & 0 & \dots & \alpha_d \\ \alpha_1 & \beta_2 & \alpha_2 & \ddots & \vdots \\ 0 & \alpha_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \alpha_{d-1} \\ \alpha_d & \dots & 0 & \alpha_{d-1} & \beta_d \end{bmatrix}, A(Y) = \begin{bmatrix} 0 & -\alpha_1 & 0 & \dots & \alpha_d \\ \alpha_1 & 0 & -\alpha_2 & \ddots & \vdots \\ 0 & \alpha_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\alpha_{d-1} \\ -\alpha_d & \dots & 0 & \alpha_{d-1} & 0 \end{bmatrix}.$$

The numerical example considered here is an 11-particle periodic Toda lattice taken from [21] with the initial conditions

$$q(0) = [0, \dots, 0]^T,$$

$$p_j(0) = \begin{cases} 4, & 1 \le j \le 4, \\ 0, & j \ge 5. \end{cases} \tag{5.6}$$

The periodic Toda lattice is not asymptotically free and has considerably complicated motion. Figure 1 shows the position and momenta of the 11 particles up to $t_{final} = 10.0$.

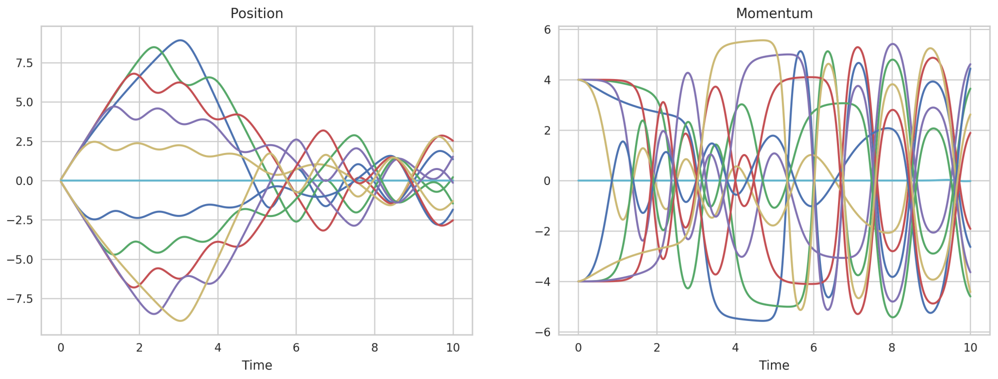


FIG. 1. *Symmetric periodic* 11*-particle Toda lattice solutions with initial conditions as in Eq.* (5.6).

5.1.1. Simulation parameters. The following numerical experiments are all performed on this 11-particle periodic Toda lattice with initial conditions as in (5.6) and a fixed $t_{final} = 10.0$. A Picard iteration tolerance of 10^{-12} is used on the maximum absolute value of the residual at the end of the time step. The reference solution is taken as the Leg-6 method with $\Delta t = 2^{-15} t_{final}$ or 32768 steps. The reported error is defined as the matrix 2-norm of the absolute value of the difference between the solution in question and the reference solution.

The methods have been implemented using the *LibPFASST*[1] library. Communication between pipelined iterations is done in *LibPFASST* using MPI, and the variable N_P corresponding to the number of parallel steps is also the number of MPI ranks. For parallelization across the method, OpenMP is used to parallelize the steps in the nonlinear iteration. All timing results were performed on a single compute node of NERSC's Cray XC30 supercomputer, Edison, which contains 24 hardware cores and 64 GB of memory.

5.1.2. Serial results. We first perform convergence tests to examine the error with respect to Δt for different Magnus methods. Figure 2 shows the behavior for each method summarized in Table 2. Each method displays the proper convergence rate for a range of Δt. For the second-order methods, note that the error for Leg-2 is significantly smaller than that of Lob-2, which implies the dominant error term for Lob-2 is due to the quadrature rule as opposed to the truncation of the Magnus expansion. The Leg-2 method requires more serial work in this case due to the use of three quadrature nodes instead of two. The difference between the fourth-order methods is less significant. Leg-4-3 is more accurate than Lob-4-1 (with a higher serial cost), but since the main difference between the two methods is how the second term in the Magnus expansion is treated, the difference between the two is smaller than for the second-order methods. Leg-6 is clearly more accurate than the other methods. Note that only about nine significant digits of accuracy is attainable for the reference solution for this problem using double precision due to the sensitivity of the solution to perturbations.

To better demonstrate the relative computational cost of each method, Figure 3 shows the total serial wall-time versus number of steps for the experiment above. As expected, Lob-2 is the method with the shortest time to solution. Lob-4-1, the

[1] *LibPFASST* is available at https://github.com/libpfasst/LibPFASST.

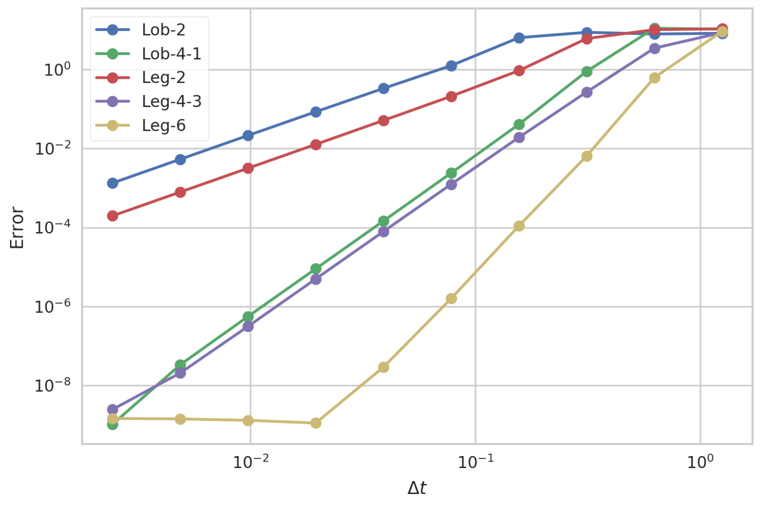


FIG. 2. *Error at* $t_{final} = 10.0$ *versus* Δt *for the Toda lattice test case.*

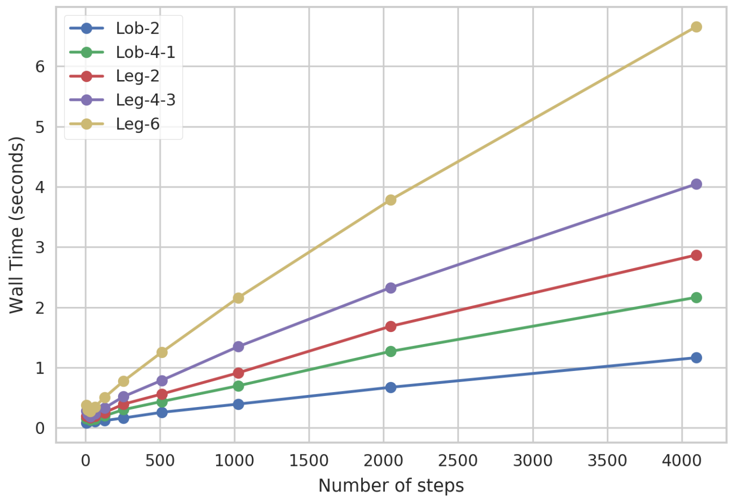


FIG. 3. *Total wall-time for the solution for the Toda lattice test case for fixed* $t_{final} = 10.0$.

simplest fourth-order method that can be constructed, is actually less expensive than the second-order Leg-2 scheme despite the fact that Lob-4-1 requires a matrix commutator. For the small problem size used here, the matrix commutators are relatively inexpensive, and the fact that matrix exponentials must be computed at three internal quadrature nodes for Leg-2 more than makes up for the lack of commutator terms. Leg-4-3 and Leg-6 are unsurprisingly the most expensive in serial.

Note that the cost of the methods displayed in Figure 3 does not grow exactly linearly with the number of time steps. This is due to the fact that the number of Picard iterations needed to converge to the tolerance depends on the time step Δt. Figure 4 shows the average number of iterations over all time steps for each method as a function of Δt. Note the higher-order methods require moderately fewer iterations than lower-order methods, and as the time step gets larger, the number of iterations required for convergence grows rapidly.

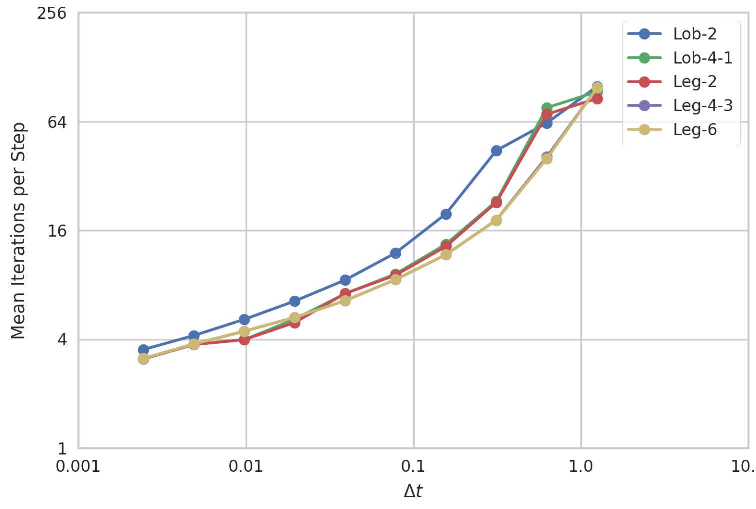


FIG. 4. *Average number of iterations as a function of Δt for serial methods on the Toda lattice test case.*

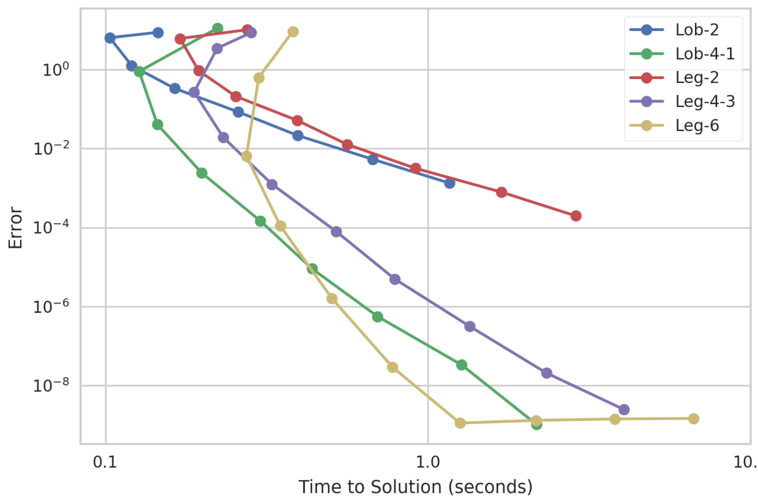


FIG. 5. *Error versus time to solution for the serial Toda lattice test case. Each point on a line, read from top to bottom, represents a twofold increase in the number of steps.*

The three figures provided above demonstrate that it is not necessarily trivial to choose a method and time step that will provide a solution to a given accuracy with the least computational effort. To illustrate this, Figure 5 shows the accuracy versus wall-clock time for the Toda lattice test. Reading from left to right for a given accuracy shows in increasing order, the methods with the fastest time to solution. While Lob-2 is by far the cheapest method, it is only the fastest method for simulations where the error is $O(1)$. Lob-4-1 is the most efficient for error tolerances to about 10^{-6}, after which Leg-6 becomes the most efficient. For an error of about 10^{-6}, Leg-6 and Lob-4-1 are more than an order of magnitude more efficient than the second-order methods.

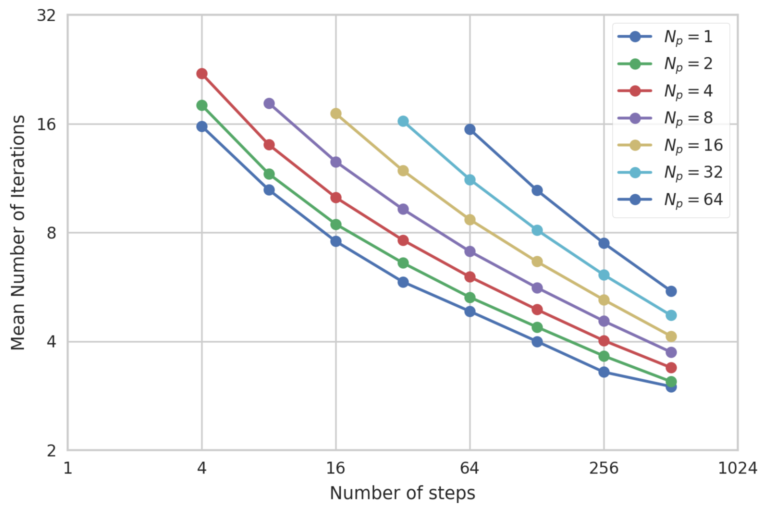
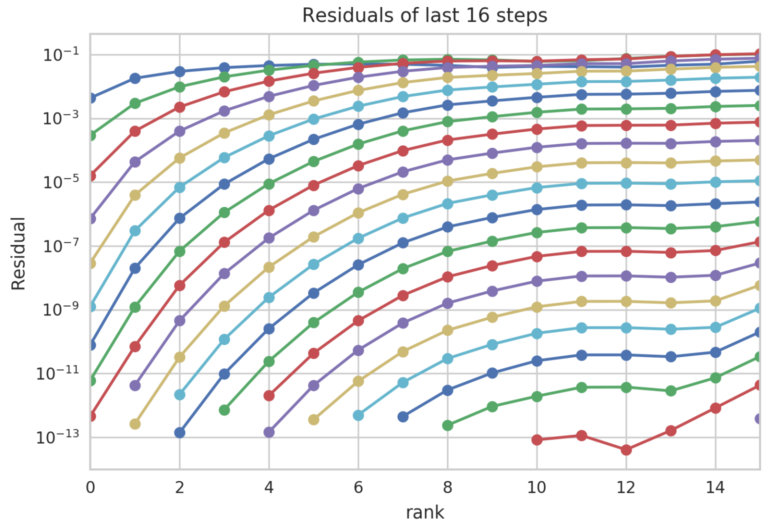


FIG. 6. *Average number of pipelined iterations to reach residual of* 10^{-12} *for the parallel Toda lattice test case using the Leg-*6 *method.*

5.1.3. Parallel-in-time results. In this section, the relative performance of parallel Magnus integrators is explored. We first consider the speedup due to parallelization over time steps by pipelining the Picard iterations. As discussed in section 4, the theoretical speedup from pipelining is bounded by the number of serial iterations required for the method and depends on how the total number of parallel iterations required to reach convergence grows as the number of time-parallel steps is increased. As in the serial case, the number of iterations required depends on the time step but now depends also on the number of parallel time steps in the pipeline. Figure 6 demonstrates this dependence for the Toda lattice test using method Leg-6 by plotting the average number of iterations required for convergence for different Δt and parallel steps, denoted by N_P. As in the serial case, the number of iterations required decreases with decreasing Δt and here it also increases for fixed Δt as N_P increases.

A second way to display the convergence behavior of the pipelined iteration is to plot the residual after each iteration for each time rank. Figure 7 shows this data for the Leg-6 method using a time step of $\Delta t = 10/128$ in the top panel and $\Delta t = 10/1024$ in the bottom panel for 16 parallel time steps. As discussed in section 4.2, the speedup achievable from pipelining depends on the ratio K_P/N_P. In this example it is clear that K_P/N_P is decreasing for large N_P as more pipelined time steps are used.

The nontrivial dependence of the parallel iterates on both Δt and N_P makes it difficult to predict which method with which parameters will minimize the time to solution for a given accuracy. As in the serial case, it is instructive to consider the accuracy versus wall-clock for different methods with different number of time-parallel steps. Figure 8 displays this information for each of the methods with increasing numbers of processors. Across a single method, e.g., Leg-6, there's an optimal value of parallelization due to the fact that the increased number of iterations required in the pipeline algorithm starts to cost more than it gains. It is also likely for this test case that the small problem size implies that communication latency is not negligible. Nevertheless, it is again clear that the higher-order parallel method gives a shorter time to solution than lower-order alternatives. The second-order methods are only less expensive than the higher-order methods when no digits of accuracy are computed.



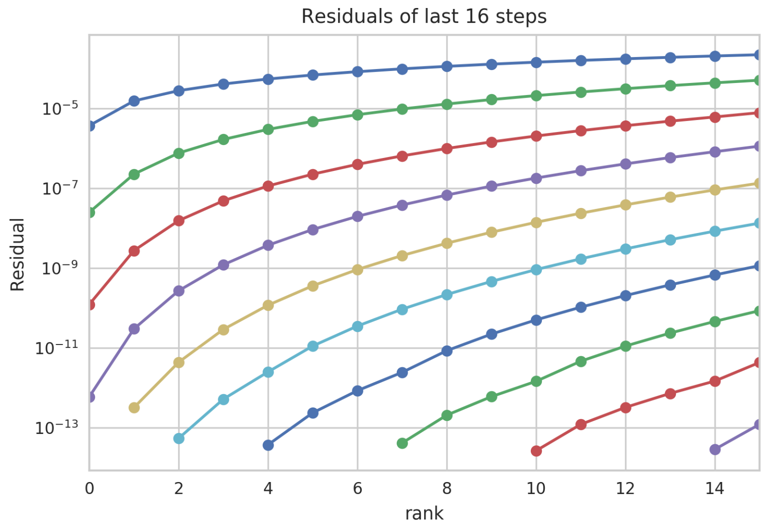


FIG. 7. *Residual convergence for each processor in a* 16 *MPI task simulation using the second-order Legendre method Leg-*6 *with* 128 *steps (above) and* 1024 *steps (below). Every reoccurrence of a given color is another* 6 *iterations.*

Finally, we present preliminary results using parallelization across the method and across the time steps. Figure 9 shows the error versus time to solution for the serial implementation and time- and method-parallel methods. The particular configuration of $N_p = 8$ and three OpenMP threads uses the entirety of one hardware node on NERSC's Edison Cray XC30 supercomputer with no hyperthreading. The additional parallel across method provides an additional factor of at best two using simple OpenMP parallel do loops over nodes. Using both types of parallelism in this context gives up to a factor of 4.7 in the overall compute time compared to serial methods using 24 total processors. Note that compared to serial second-order methods, the parallel sixth-order method can compute a solution in less time with more than four orders of magnitude lower error.

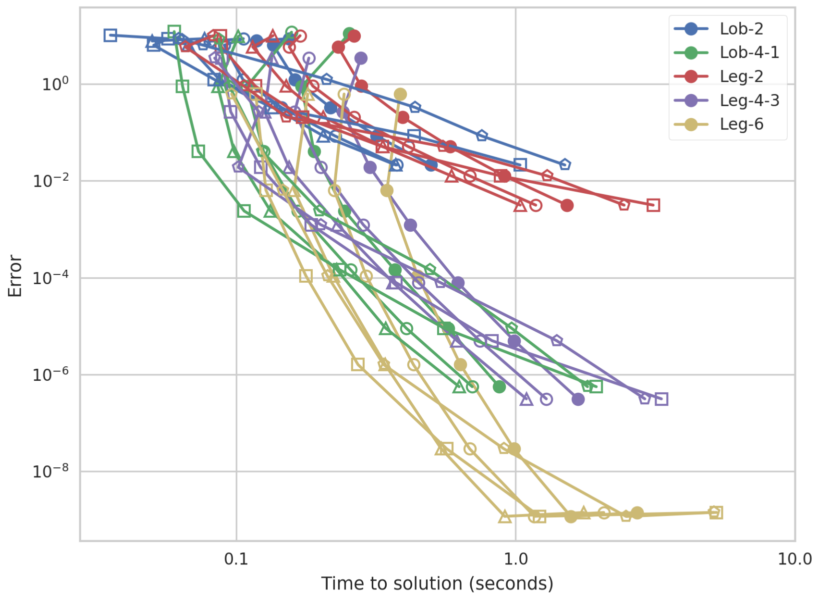


FIG. 8. *Error versus time to solution for an 11-particle periodic Toda lattice example. Each color represents a different method and each marker represents a different number of time-parallel steps, $N_p = 1, 2, 4, 8, 16$. Closed circles represent the serial computation, open circles $N_p = 2$, triangles $N_p = 4$, squares $N_p = 8$, and pentagons $N_p = 16$.*

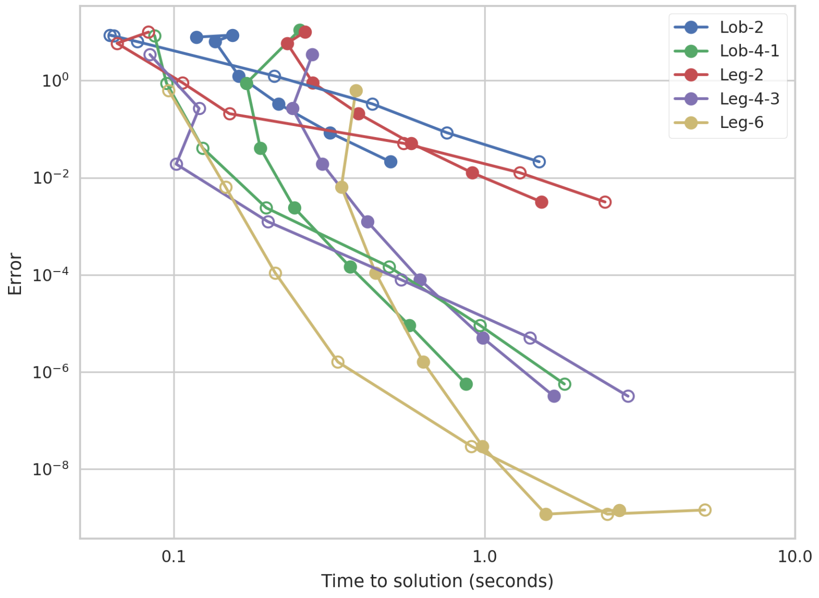


FIG. 9. *Error versus time to solution for an 11-particle periodic Toda lattice example. Solid markers indicate serial calculations and open markers indicate time- and method-parallel runs with eight MPI tasks and three OpenMP threads.*

5.2. Test Problem 2: An RT-TDDFT proxy. The movitating application for the parallel Magnus methods introduced here is the simulation of electron dynamics using RT-TDDFT. In this section we study the parallel performance of the time-parallel Magnus methods on a test problem with a structure similar to RT-TDDFT but simpler.

5.2.1. Physical background. RT-TDDFT describes the time evolution of the density matrix P through the von Neumann equation of motion

$$P' = -i[F(P), P], \tag{5.7}$$

where $F(P)$ is the Fock matrix. The size of the matrices N scales with the number of electrons times the number of basis functions used to represent the electron density. The evaluation of $F(P)$, the construction of the Fock matrix, requires for each entry in P the approximation of one- and two-electron integrals which formally requires $O(N^4)$ work. Furthermore, the time scales on which electron dynamics occur are typically on the order of tens or hundreds of atto-seconds, hence even simulations of moderate sized molecules at the femto-second range are extremely computationally expensive. In the RT-TDDFT simulation suite in the NWChem code [20], (5.7) is integrated using a serial second-order Magnus integrator, and part of the motivation of this paper is to improve the efficiency of this choice.

5.2.2. The proxy problem. To give an example of the performance of the parallel Magnus methods without requiring the considerable infrastructure necessary to compute the true Fock matrix, we consider a test problem inspired by a simplified one-dimensional RT-TDDFT problem. Here P is again an $N \times N$ complex matrix, and

$$F(P)_{i,j} = \sum_{i=1}^{N}\sum_{j=1}^{i}\left(Zf_1(i,j) + \sum_{m=1}^{N}\sum_{n=1}^{N} E\left(f_2(i,j,m,n) - \frac{1}{2}f_2(i,n,m,j)\right)\right). \tag{5.8}$$

The functions f_1 and f_2 correspond to the one- and two-electron integrals in RT-DTDFT, with the f_1 term corresponding to the single electron Hamiltonian and the two f_2 terms corresponding to the Coulomb and exchange operators, respectively. The simplified forms here are derived by considering single Gaussian basis functions in one dimension with single point approximations of the integrals. Specifically

$$f_1(i,j) = -\frac{P_{i,j}\bar{P}_{j,i}}{r^2}\exp\left(-\frac{(x(i)-x(j))^2}{2}\right) \tag{5.9}$$

with $r = (x(i) + x(j))/2$. Similarly

$$f_2(i,j,m,n) = \frac{P_{m,n}\bar{P}_{n,m}P_{i,j}\bar{P}_{j,i}}{r^2}\exp\left(-\frac{(x(i)-x(j))^2}{2}\right)\exp\left(-\frac{(x(m)-x(n))^2}{2}\right) \tag{5.10}$$

with $r = ((x(i) + x(j))/2 - (x(m) + x(n))/2)$. Note that the computation of $F(P)$ in the problem is formally $O(N^4)$, whereas the matrix commutator terms are $O(N^3)$. The matrix exponential when computed by power series is also $O(PN^3)$, where P is the number of terms in expansion. Hence in this application (as in full RT-TDDFT simulations), the computation of $F(P)$ is the dominant cost for even moderate N. In the numerical tests, the spatial locations are uniformly spaced on $[-1, 1]$ so that $x(i+1) - x(i) = 2/(N-1)$. The values for Z and E are 0.5 and 0.05, respectively.

5.2.3. Parallel-in-time results. For the test problem described above, we run the parallel Magnus algorithms Lob-2, Lob-4-1, and Leg-6 on a 20-particle RT-TDDFT proxy problem as described above, using a final time of the run $T = 0.05$.

For each choice of method, runs using 48, 96, 144, and 192 time steps are compared for different numbers of parallel processors $N_p = 1, 2, 4, 8, 16$, and 24. A uniform time step is used in all cases, which is justified since the character of the dynamics does not change in time. As above, computations are run on one hardware node on NERSC's Edison Cray XC30 consisting of 24 cores. For these comparisons, only the parallelism across time steps is used. Unlike the results in the previous section, the convergence tolerances are adjusted by case to avoid unnecessary iterations. In practice this means that the tolerances are larger for less accurate simulations since additional SDC iterations will reduce the residual but not the numerical error determined by the underlying quadrature rule. The tolerances here were chosen using knowledge of the error in the simulations and are shown in Table 3. Dynamically choosing the optimal tolerance for serial or parallel SDC methods is an open problem.

Figure 10 compares the accuracy versus total computational time for each variation and demonstrates the attractiveness of both higher-order methods and time parallelism. The reported error in each case is the maximum absolute error along the diagonal of P (which in RT-TDDFT are the relevant quantities). It is clear from the timings that significant parallel speedup is attainable for methods of all order.

TABLE 3
Residual tolerances for the RT-TDDFT proxy problem for different methods and number of steps.

	48	96	144	196
Lob-2	$1e-4$	$1e-4$	$2e-5$	$1e-5$
Lob-4-1	$1e-5$	$1e-6$	$5e-7$	$1e-7$
Leg-6	$1e-6$	$1e-8$	$1e-9$	$1e-10$

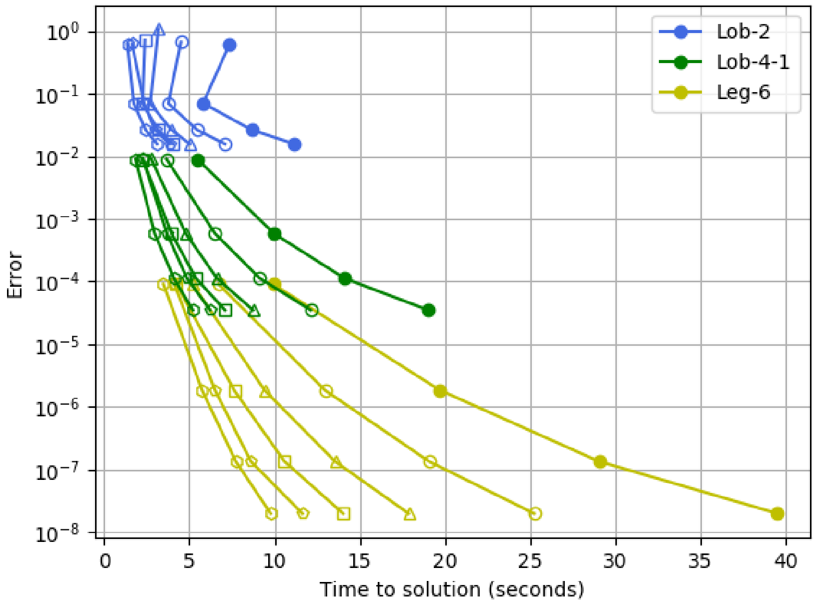


FIG. 10. *Error versus time to solution for a 20-particle RT-TDDFT proxy example. Each color represents a different method and each marker represents a different number of time-parallel steps, $N_p = 1, 2, 4, 8, 16, 24$. Closed circles represent the serial computation, open circles $N_p = 2$, triangles $N_p = 4$, squares $N_p = 8$, pentagons $N_p = 16$, hexagons $N_p = 24$.*

As expected, the smaller the error tolerance, the larger the possible parallel speedup since there are in general more iterations to amortize over processors. Note that serial second-order methods are more expensive than the fastest fourth-order methods. In particular, the fastest serial fourth-order method runs in less time than the fastest serial second-order method because fewer iterations are needed for convergence (despite the smaller residual tolerance). Parallel sixth-order methods run faster than serial second-order methods while producing smaller errors by several orders of magnitude. In general, the optimal choice of method and parallelization depends on the error tolerance.

6. Discussion.

Much of the initial work on Magnus integrators is focused on linear problems where only a single evaluation of the Magnus expansion is required for each time step. In contrast, this paper explores the accuracy and cost of Magnus integrators applied to nonlinear problems. Nonlinear Magnus methods require solving an implicit equation involving the Magnus expansion to obtain the solution in each time step, and the methods proposed here use a simple fixed point iteration for this solution that can be readily parallelized in time.

One conclusion presented here is that in both the linear and nonlinear cases, straightforward parallelization across the method is possible, leading to higher-order methods with only marginally higher computational cost than lower-order methods. This is complimentary to previous results in the literature where considerable effort is placed on reducing the complexity of each of the terms in regards to the number of commutators required.

The second level of parallelism described in this work, namely parallelization across the time steps through pipelined iterations, can further decrease the overall time to solution for nonlinear problems. There is a nontrivial relationship between the number of pipelined time steps, the time step size, and the number of iterations required for convergence of the iteration, hence it is not easy to predict a priori which choice of parameters will lead to the shortest wall-clock time given a desired level of accuracy. Nevertheless, for the test cases considered here, in most situations, the parallel sixth-order methods require the least computation time and are far superior to serial second-order methods.

It is important to note that the results here are preliminary and are performed on two test cases of moderate size. In general, the possible parallel speedup attainable for a given problem will depend on the sensitivity of the problem to perturbations, the relative cost of the operations such as computing commutators or the matrix exponential, and the ratio of computation to communication costs. In future work, the authors will use this parallel methodology to investigate real-time electronic dynamics, where the calculation of the right-hand-side values is more expensive than both commutators and matrix exponentials as in the second numerical example. This paper addresses only the use of time parallelism to reduce the wall-clock time of serial Magnus methods, and not important issues like the best way to compute the matrix exponential, the optimal choice of time step, nor the dynamic stopping criteria for iterations. Such issues are present in both serial and parallel implementations of Magnus methods and are most likely problem dependent.

Appendix A. Quadrature rules for intermediate nodes.

The necessary weights for implementing each of the methods Lob-4-1, Leg-2, Leg-4-3, and Leg-6 for nonlinear problems are presented here. For Lob-2, no additional rules are necessary.

For the computation of the single integrals in the first Magnus term, the necessary rules take the form

$$\Omega_m^{(1)} = \Delta t \sum_{j=1}^{M} q_{m,j}^{(1)} A_j, \tag{A.1}$$

and the coefficients $q_{m,j}^{(1)}$ correspond to the classical collocation schemes (see, e.g., [9]) For Lob-3, only one additional rule is needed at the midpoint t_2,

$$q_{2,j}^{(1)} = \left[\frac{5}{24}, \frac{1}{3}, -\frac{1}{24}\right]. \tag{A.2}$$

For Leg-3, rules for each node are required, and the $q_{m,j}^{(1)}$ are given by

$$q_{m,j}^{(1)} = \begin{bmatrix} \frac{5}{36} & \frac{2}{9} - \frac{\sqrt{15}}{15} & \frac{5}{36} - \frac{\sqrt{15}}{30} \\ \frac{5}{36} + \frac{\sqrt{15}}{24} & \frac{2}{9} & \frac{5}{36} - \frac{\sqrt{15}}{24} \\ \frac{5}{36} + \frac{\sqrt{15}}{30} & \frac{2}{9} + \frac{\sqrt{15}}{15} & \frac{5}{36} \end{bmatrix}. \tag{A.3}$$

For the second term $\Omega^{(2)}$ there are two versions described in the linear case by (3.2) and (3.3). In the first case, only one additional coefficient is needed, namely $q_2^{(2)-1} = 1/48$. In the second case, we have

$$\Omega_m^{(2)-3} = q_{m,1}^{(2)-3}[A_1, A_2] + q_{m,2}^{(2)-3}[A_1, A_3] + q_{m,3}^{(2)-3}[A_2, A_3], \tag{A.4}$$

with the values of $q_{m,j}^{(2)-3}$ given by

$$q_{m,j}^{(2)-3} = \begin{bmatrix} -7.0825623244174\text{e}{-4} & -3.5291589565775\text{e}{-2} & -7.8891497044705\text{e}{-2} \\ 2.0142743933468\text{e}{-4} & 4.4826196136660\text{e}{-3} & -1.8131905893999\text{e}{-2} \\ -2.6081558162830\text{e}{-6} & -5.6936734355286\text{e}{-4} & -3.5152700676886\text{e}{-2} \end{bmatrix}. \tag{A.5}$$

The third term takes the form

$$\begin{aligned} \Omega_m^{(3)} = & \left[q_{m,1,1}^{(3)} A_1 + q_{m,1,2}^{(3)} A_2 + q_{m,1,3}^{(3)} A_3, [A_1, A_2]\right] \\ & + \left[q_{m,2,1}^{(3)} A_1 + q_{m,2,2}^{(3)} A_2 + q_{m,2,3}^{(3)} A_3, [A_1, A_3]\right] \\ & + \left[q_{m,3,1}^{(3)} A_1 + q_{m,3,2}^{(3)} A_2 + q_{m,3,3}^{(3)} A_3, [A_2, A_3]\right] \end{aligned} \tag{A.6}$$

with

$$q_{1,i,j}^{(3)} = \begin{bmatrix} 1.4667828928181\text{e}{-6} & -2.5468454487434\text{e}{-6} & 7.1885579589404\text{e}{-7} \\ -3.0653702506833\text{e}{-7} & 6.9623363228690\text{e}{-7} & -1.9684558120029\text{e}{-7} \\ -2.2622163607144\text{e}{-8} & -2.7279719400850\text{e}{-9} & 8.5484354192049\text{e}{-10} \end{bmatrix}, \tag{A.7}$$

$$q_{2,i,j}^{(3)} = \begin{bmatrix} 1.0401143365317\text{e}{-3} & -1.7143302808715\text{e}{-3} & 1.9808827525182\text{e}{-4} \\ -6.9105495969459\text{e}{-5} & 2.9054016014502\text{e}{-4} & -3.4658846939476\text{e}{-5} \\ 9.2451884893203\text{e}{-5} & 1.2595057164957\text{e}{-5} & -2.4709074423914\text{e}{-6} \end{bmatrix}, \tag{A.8}$$

(A.9)

$$q^{(3)}_{3,i,j} = \begin{bmatrix} 4.1482959753609\text{e}{-3} & -6.3874218931689\text{e}{-3} & -3.5942319108173\text{e}{-3} \\ 9.9737811032708\text{e}{-4} & 1.2415302375576\text{e}{-4} & -3.8059754231607\text{e}{-4} \\ 3.7183849345731\text{e}{-3} & 1.6935142950568\text{e}{-3} & -1.0604085845381\text{e}{-3} \end{bmatrix}.$$

Finally, for the fourth term, $\Omega_m^{(4)}$ is computed as in (3.7) and (3.8), with $q^{(4)} = 1/60$ and $q_j^{(1)}$ in (3.8) replaced with $q_{m,j}^{(1)}$ from (A.3).

REFERENCES

[1] S. Blanes, F. Casas, J. A. Oteo, and J. Ros, *The Magnus expansion and some of its applications*, Phys. Rep., 470 (2009), pp. 151–238.

[2] S. Blanes, F. Casas, and J. Ros, *Improved high order integrators based on the Magnus expansion*, BIT, 40 (2000), pp. 434–450.

[3] S. Blanes and P. C. Moan, *Fourth-and sixth-order commutator-free Magnus integrators for linear and non-linear dynamical systems*, Appl. Numer. Math., 56 (2006), pp. 1519–1537.

[4] K. Burrage, *Parallel methods for ODEs*, Adv. Comput. Math., 7 (1997), pp. 1–3.

[5] L. Euler, *Institutiones calculi integralis*, v. 2, Academiae Imperialis Scientiarum, 1768.

[6] M. J. Gander, 50 *years of time parallel time integration*, in Multiple Shooting and Time Domain Decomposition Methods, T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, eds., Springer, Berlin, 2015, pp. 69–113.

[7] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer, Berlin, 2006.

[8] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations* I. *Non-stiff Problems*, Math. Comput. Simul. 29, Springer, Berlin, 1987.

[9] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations* II*: Stiff and Differential-Algebraic Problems*, Springer, Berlin, 1991.

[10] N. J. Higham, *The Scaling and Squaring Method for the Matrix Exponential Revisited*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 1179–1193.

[11] M. Hochbruck and A. Ostermann, *Exponential integrators*, Acta Numer., 19 (2010), pp. 209–286.

[12] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna, *Lie-group methods*, Acta Numer., 9 (2000), pp. 215–365.

[13] A. Iserles and S. P. Nørsett, *On the solution of linear differential equations in Lie groups*, Philosophical Transactions: Mathematical, Physical and Engineering Sciences, 357 (1999), pp. 983–1019.

[14] W. Magnus, *On the exponential solution of differential equations for a linear operator*, Communications on pure and applied . . . , VII (1954), pp. 649–673.

[15] C. Moler and C. Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later*, SIAM Review, 45 (2003), pp. 3–49.

[16] H. Munthe-Kaas and B. Owren, *Computations in a free Lie algebra*, Philos. Trans. A, 357 (1999), pp. 957–981.

[17] J. Nievergelt, *Parallel methods for integrating ordinary differential equations*, Commun. ACM, 7 (1964), pp. 731–733.

[18] M. Thalhammer, *A fourth-order commutator-free exponential integrator for nonautonomous differential equations*, SIAM J. Numer. Anal., 44 (2006), pp. 851–864.

[19] M. Toda, *Vibration of a chain with nonlinear interaction*, J. Phys. Soc. Japan, 22 (1967), pp. 431–436.

[20] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, D. W. H. J. J. van Dam, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, *NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations*, Comput. Phys. Commun., 181 (2010), pp. 1477–1489.

[21] A. Zanna, *Numerical Solution of Isospectral Flows*, Ph.D. thesis, University of Cambridge, 1998.