Comparative performance of exponential, implicit, and explicit integrators for stiff systems of ODEs

J. Loffeld\textsuperscript{a}, M. Tokman\textsuperscript{a,*}

\textsuperscript{a}School of Natural Sciences, University of California, 5200 N. Lake Road, Merced, CA 95343

Abstract

Exponential integrators have enjoyed a resurgence of interest in recent years, but there is still limited understanding of how their performance compares with state-of-art integrators, most notably the commonly used Newton-Krylov implicit methods. In this paper we present comparative performance analysis of Krylov-based exponential, implicit and explicit integrators on a suite of stiff test problems and demonstrate that exponential integrators have computational advantages compared to the other methods, particularly as problems become larger and more stiff. We argue that the faster convergence of the Krylov iteration within exponential integrators accounts for the main portion of the computational savings they provide and illustrate how the structure of these methods ensures such efficiency. In addition, we demonstrate computational advantages of newly introduced [17] exponential propagation Runge-Kutta (EpiRK) fifth order methods. The presented detailed analysis of the methods’ performance provides guidelines for construction and implementation of efficient exponential methods and the quantitative comparisons instruct selection of appropriate schemes for other problems.

Keywords: exponential integrators, Krylov projections, EPIRK methods, stiff systems, large scale computing

1. Introduction and background

Scientific problems are often cast in the form of initial-value problems for very large systems of ordinary differential equations (ODEs). The numerical integration of these large systems can be very computationally demanding, so it is desirable that integrators for these problems be as efficient as possible. We are interested in which types of integrators are most computationally efficient for systems which are very large, stiff, and in general have a nonsymmetric Jacobian.

\textsuperscript{*}Corresponding author. Tel.: +1 209 349 2857; fax: +1 209 228 4060.

Email addresses: jloffeld@ucmerced.edu (J. Loffeld), mtokman@ucmerced.edu (M. Tokman)

Preprint submitted to Elsevier December 16, 2011
Stiff systems often preclude the use of explicit integrators since these methods require the integration step size to be very small to ensure numerical stability [1, 2]. Instead, stiff problems are usually solved using implicit integrators. Such methods require solutions of an implicit system each integration step. For general non-symmetric problems the Newton iteration is typically used to solve the system. For very large problems, direct methods for solving the linear system within each Newton iteration are computationally infeasible and iterative methods are typically employed. Modern iterative methods are based on the Krylov iteration and currently the most common type of algorithms for solving large stiff problems are the Newton-Krylov implicit integrators [3]. The efficiency of Newton-Krylov methods is often predicated upon the construction of an effective preconditioner to solve the linear systems. However, constructing such a preconditioner can be very difficult and highly problem dependent. Frequently one wants to avoid building a preconditioner and to use the most efficient black-box time integrator. In this paper, we limit ourselves to considering those problems where an efficient preconditioner is not available.

Recently, exponential integrators have emerged as a potential alternative class of methods for efficiently solving large stiff problems. When first introduced, exponential methods had been considered computationally unattractive due to the high cost of evaluating the exponential functions of large matrices that arise in these methods. However, the methods started to draw attention when the use of Krylov projection techniques allowed these matrix exponential terms to be evaluated efficiently [4, 5]. Since then, a number of exponential integrators for general stiff systems have been proposed [6, 7, 8, 9, 10, 11, 12].

Despite the surge of interest in exponential integrators, there is still only limited understanding of how exponential integrators perform on large scale problems, particularly in comparison to Newton-Krylov implicit integrators. Partially as a result of this, exponential methods have not been widely used. This paper presents comparative performance analysis of Krylov-based exponential integrators, Newton-Krylov implicit integrators and an explicit method on a suite of stiff test problems. We show that exponential schemes compare well to currently used methods, particularly as problems scale to larger size. We also examine how the structure of the exponential integrators allows them to outperform Newton-Krylov implicit methods and illustrate these ideas with numerical experiments. Finally, we then demonstrate how the structure of a method translates to efficiency by presenting the first numerical study of a newly introduced [13] fifth order three-stage exponential propagation iterative (EpiRK) method designed to optimize performance.

The paper is organized as follows. Section 2 provides a description of the structure of Krylov-based exponential methods and contrasts them with Newton-Krylov implicit integrators. Section 3 describes the methods and problems used for the analysis, and Section 4 presents the results of the numerical experiments and explains the performance differences between the methods. Section 5 presents numerical tests of the optimized three-stage fifth order exponential integrator and comparisons with other methods. Variable step size implicit and exponential integrators are compared in Section 6. Finally, Section
2. Structure of exponential integrators

In this section we provide a brief introduction to the derivation and structure of exponential integrators and highlight the elements which have the most impact on their computational performance, particularly in contrast to Krylov-based implicit integrators. Consider the initial-value problem for a nonlinear autonomous system of ODEs

\[
\frac{dy}{dt} = F(y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^N,
\]

where \( N \) is large and the system is stiff. There is no loss of generality in considering an autonomous system since a non-autonomous one can be converted to autonomous form by adding the equation \( t' = 1 \).

To derive an exponential method, we first rewrite equation (1) using a Taylor expansion as

\[
\frac{dy(t + \Delta t)}{dt} = F(y(t)) + J(y(t)) [y(t + \Delta t) - y(t)] + R(y(t + \Delta t)),
\]

where \( J = J(y(t)) = D_y F(y(t)) \) is the Jacobian of \( F(y(t)) \), which is assumed to exist, and the nonlinear remainder function \( R(y(t)) \) is defined as

\[
R(y(t + \Delta t)) = F(y(t + \Delta t)) - F(y(t)) - J(y(t)) [y(t + \Delta t) - y(t)].
\]

Using the integrating factor \( e^{\Delta t J} \) we obtain the integral form of the equation

\[
y(t + \Delta t) = y(t) + (e^{\Delta t J} - I)(J \Delta t)^{-1} \Delta t F(y(t)) + \int_0^1 e^{J \Delta t (1-\theta)} R(y(t + \Delta t \theta)) d\theta.
\]

Equation (3) is a starting point for the derivation of most exponential methods. An exponential method is constructed from (3) by numerically approximating the integral term. For example, a two-node Runge-Kutta-type quadrature for the integral results in a second-order two-stage scheme

\[
\begin{align*}
   r_1 &= y_n + h \varphi_1 \left( \frac{1}{2} h J_n \right) F_n, \\
   y_{n+1} &= y_n + h \varphi_1 \left( h J_n \right) F_n + \frac{3}{2} h \varphi_2 \left( h J_n \right) R(r_1),
\end{align*}
\]

where \( F_n = F(y(t_n)) \), \( J_n \) is the Jacobian of \( F(y(t)) \) evaluated at \( t_n \), \( h = \Delta t \), \( \varphi_1(z) = (e^z - 1)/z \) and \( \varphi_2(z) = (e^z - z - 1)/z^2 \).

In general, approximating the integral using a polynomial expansion will result in an exponential integrator composed of linear combinations of products of \( \varphi \)-functions

\[
\varphi_k(z) = \int_0^1 e^{z(1-\theta)} \frac{\theta^{k-1}}{(k-1)!} d\theta, \quad k = 0, 1, 2, \ldots
\]
acting on vectors, i.e.
\[ \varphi_0(hJ)b_0 + \varphi_1(hJ)b_1 + \varphi_2(hJ)b_2 + \cdots + \varphi_i(hJ)b_i, \]
where \( b_i \in \mathbb{R}^N \) and \( J \in \mathbb{R}^{N \times N} \). The matrix valued analytic functions \( \varphi_n(hJ) \) can be defined via Taylor series as
\[ \varphi_i(hJ) = \sum_{k=0}^{\infty} \frac{(hJ)^i}{(i+k)!} \]
(5)

Since \( N \) is large, computing the products of the \( \varphi \)-functions and vectors (e.g. the \( \varphi_1(hJn)F_n \) term in (4)) by algorithms such as Taylor or Padé approximations is computationally prohibitively expensive [15]. Thus we turn to Krylov algorithms which approximate such products by projections onto the Krylov subspace \( K_m(hJ,b) = \text{span}\{b, (hJ)b, (hJ)^2b, \ldots, (hJ)^{m-1}b\} \).

For general nonsymmetric \( J \), Krylov projection is done using the Arnoldi iteration [14]. The Arnoldi algorithm employs a modified Gram-Schmidt process to produce a matrix \( V_m \) with column vectors forming an orthonormal basis of the Krylov subspace, which in turn gives the orthogonal projection matrix \( V_mV_m^T \).

The upper Hessenberg matrix
\[ H_m = V_m^T(hJ)V_m \]
(6)
is obtained as a side product of the iteration. The product of a matrix function \( f(hJ) \) and a vector \( b \) is approximated by using the projection matrix \( V_mV_m^T \) as
\[ f(hJ)b \approx V_mV_m^Tf(hJ)V_mV_m^Tb. \]
(7)
\( H_m \) is then used to evaluate
\[ V_m^Tf(hJ)V_m \approx f(H_m), \]
(8)
producing the final approximation
\[ f(hJ)b \approx V_mf(H_m)V_m^Tb. \]
(9)
Since the first column vector of \( V_m \) is \( v_1 = b/\|b\|_2 \), we can use \( V_m^Tb = \|b\|_2e_1 \) to simplify (9) as
\[ f(hJ)b \approx \|b\|_2V_mf(H_m)e_1. \]
(10)
\( H_m \) is expected to be small (\( m \ll N \)), so computing this approximation is considerably cheaper than evaluating \( \varphi(hJ)b \) directly and can be done using the algorithms such as Taylor or Padé approximations [15].

The computational cost of performing this iteration is determined by how rapidly the Krylov iteration converges. In general, the rate depends on the eigenvalues of \( J \), the type of function \( f \), and the magnitudes of \( h \) and \( b \). The bound derived in [16] showed that in the case where \( f = \varphi_0 \) and \( J \) is a Hermitian
negative semi-definite matrix the convergence becomes superlinear when 
\( m \geq \sqrt{\|hJ\|} \). But in general determining the rate of convergence is theoretically
difficult since it depends on the spectrum of \( J \) and fast convergence is often
observed even for smaller \( m \). In the next section, we argue that exponential
functions should converge more quickly in the Krylov iteration compared to
Newton-Krylov implicit methods due to the choice of \( f \) used in the two methods.
In section 4, we support this claim with numerical experiments.

We now provide examples of exponential integrators, and consider ways in
which the structure of an exponential method affects its efficiency. The first
example is the fourth-order method Exp4 [7]:

\[
\begin{align*}
    k_1 &= \varphi_1(\frac{1}{3}hJ_n)F_n, \quad
    k_2 = \varphi_1(\frac{2}{3}hJ_n)F_n, \quad
    k_3 = \varphi_1(hJ_n)F_n, \quad (11) \\
    w_4 &= -\frac{7}{300}k_1 + \frac{97}{150}k_2 - \frac{37}{300}k_3, \\
    u_4 &= y_0 + hw_4, \quad
    d_4 = F(u_4) - F_n - hJ_0w_4, \\
    k_4 &= \varphi_1(\frac{1}{3}hJ_n)d_4, \quad
    k_5 = \varphi_1(\frac{2}{3}hJ_n)d_4, \quad
    k_6 = \varphi_1(hJ_n)d_4, \\
    w_7 &= \frac{59}{300}k_1 - \frac{7}{15}k_2 + \frac{269}{300}k_3 + \frac{2}{3}(k_4 + k_5 + k_6), \\
    u_7 &= y_n + hw_7, \quad
    d_7 = F(u_7) - F_n - hJ_nw_7, \\
    k_7 &= \varphi_1(\frac{1}{3}hJ_n)d_7, \\
    y_{n+1} &= y_n + h(k_3 + k_4 - \frac{1}{3}k_5 + k_6 + \frac{1}{6}k_7).
\end{align*}
\]

Seven \( \varphi_1(cJ_n)v \) products have to be evaluated in this scheme. However, the
Arnoldi algorithm has a scale invariance property that can reduce the number
of necessary projections. If for a matrix \( A \) the Arnoldi algorithm yields
\( H = V^TAV \), then for the scaled matrix \( cA \) the Arnoldi algorithm gives \( cH = V^TcAV \). That means if we have computed approximation (9) as \( f(A)b \approx \|b\|_2v_mf(H_m)e_1 \), as long as the \( b \) vector remains the same and \( c < 1 \), we can
compute \( f(cA)b \approx \|b\|_2v_mf(cH_m)e_1 \) without repeating the Arnoldi iteration to
recompute \( H_m \) or \( V_m \). Noting that stages \( k_1 \) through \( k_3 \) use the same \( "b" \) vector
\( b = F_n \), and that stages \( k_4 \) through \( k_6 \) use the same vector \( b = d_4 \), we can use
this scale invariance property to compute each stage with just a single projection,
for a total of three Krylov iterations per time step. Since the \( b \) vectors
after the first stage are nonlinear remainder terms \( d_4 \) and \( d_7 \), the magnitude of the
\( b \) vectors should decrease as the approximations \( u_4 \) and \( u_7 \) becomes better
approximations to the solution. This results in fewer Krylov vectors needed to
achieve a prescribed tolerance for the projections at higher stages compared to
those of lower stages. Further, the final projection to compute \( k_7 \) scales the
Jacobian by \( \frac{1}{3} \) which should further decrease the number of Krylov vectors.

A second example is the fourth-order exponential propagation iterative scheme.
EpiRK4 [9]:
\begin{align*}
Y_1 &= y_n + a_{11} h \varphi_1 \left( \frac{1}{3} h J_n \right) F_n, \\
Y_2 &= y_n + a_{21} h \varphi_1 \left( \frac{2}{3} h J_n \right) F_n + a_{22} h \varphi_2 \left( \frac{2}{3} h J_n \right) R(Y_1), \\
y_{n+1} &= y_n + h \varphi_1 \left( h J_n \right) F_n + b_1 h \varphi_2 \left( h J_n \right) R(Y_1) \\
&\quad + b_2 h \left[ 6 \varphi_3 \left( h J_n \right) - \varphi_2 \left( h J_n \right) \right] \left( -2 R(Y_1) + R(Y_2) \right),
\end{align*}

where $s = \sqrt{30}$. As with Exp4, the scale-invariance feature of the Arnoldi iterations implies that only three Krylov projections are required per time-step. Also, similarly to Exp4, the $b$ vectors, which in this case are equal to the nonlinear remainder function $R(y)$ and its divided differences, also decrease in magnitude at higher stages. Thus with each stage the number of Krylov vectors needed for projections decreases. EpiRK4 also uses higher order $\varphi_k(z)$ functions at the higher stages which further reduces the number of Krylov vectors needed for projections at higher stages [9]. However, unlike Exp4 it does not scale down the Jacobian for the final projection, which should result in larger basis sizes for the third projection. Relative end performance is determined by how these factors balance out.

A third example is the fourth-order exponential Rosenbrock scheme ERow4 [8]:
\begin{align*}
Y_1 &= y_n + \frac{1}{2} h \varphi_1 \left( \frac{1}{2} h J_n \right) F_n, \\
Y_2 &= y_n + h \varphi_1 \left( h J_n \right) F_n + h \varphi_1 \left( h J_n \right) R(Y_1), \\
y_{n+1} &= y_n + h \varphi_1 \left( h J_n \right) F_n \\
&\quad + h \left[ 10 \varphi_3 \left( h J_n \right) - 48 \varphi_4 \left( h J_n \right) \right] R(Y_1) + h \left[ -2 \varphi_3 \left( h J_n \right) + 12 \varphi_4 \left( h J_n \right) \right] R(Y_2)
\end{align*}

The main features of this scheme are very similar to EpiRK4, in fact, the scheme can be re-written in the EpiRK form and vise versa. As in EpiRK4, higher order $\varphi$-functions are used, but in this formulation the $b$ vectors are $R(u)$ rather than the divided differences of $R(u)$. The latter feature has more of an impact on the performance of higher order schemes and the fourth order EpiRK4 and ERow4 methods have similar performance. Due to the scale invariance property, ERow4 requires three Krylov projections per time step, has reduction in the magnitude of the $b$ vectors at higher stages, uses higher-order $\varphi$-functions, but does not scale down the Jacobian for any projection. A more detailed discussion of how an exponential method’s design and structure affect its performance can be found in [17].
2.1. Comparison with Newton-Krylov implicit integrators

Modern implicit methods for large scale stiff ODE systems employ Krylov projection-based linear solvers [3]. As shown below, the relative performance of the exponential-Krylov methods and the implicit-Krylov integrators should be largely determined by the efficiency of the Krylov projections part of the algorithm. We will argue that exponential methods should have a sizable computational advantage in performing Krylov projections over implicit methods.

Implicit methods require the solution of a nonlinear algebraic system at each integration step, and this is usually accomplished using the Newton iteration [1, 2]. In the course of each iteration a linear system of the form 
\[(I - hcJ)y_{n+1} = b\] 
must be solved \((I \in \mathbb{R}^{N \times N} \) is the identity matrix and the constant coefficient \(c\) is given by a particular implicit scheme). Krylov projection techniques such as GMRES are used to evaluate what is effectively the product of a rational matrix function and a vector \((I - hcJ)^{-1}b\). This is in direct contrast with exponential methods which must evaluate the products of an exponential function and a vector \(\varphi_k(hcJ)b\). The difference in efficiency between the two classes of methods is expected to be due in part to the Krylov iterations convergence for these two types of terms. The rate of convergence of a Krylov iteration to approximate \(f(A)b\) depends on the function \(f(x)\) and the eigenvalues of \(A\). Hochbruck and Lubich derived an error bound which showed that for semi-definite Hermitian matrices, the convergence of the Krylov iteration to approximate \(f(A)b = e^Ab\) is faster than for \(f(A)b = (I - A)^{-1}b\) [16]. Similar results can be obtained for \(\varphi_k(A)b\). Error bounds for general \(A\) are difficult to obtain, but it seems reasonable to hypothesize that the rate of convergence of (10) is faster for functions where the Taylor series converges more quickly as the Krylov projection performs an orthogonal projection onto the same basis as a truncated Taylor series. Since the Taylor series for an exponential function converges faster than that of a rational function, we may expect that the Krylov iteration for exponential methods should converge more quickly than for implicit methods. Numerical evidence for this was given in [9] and we will provide further numerical support for this in Section 4.

Another major difference between exponential and implicit methods is that exponential methods need to evaluate a fixed number of Krylov projections per time-step, while Newton-Krylov implicit methods evaluate a variable number, since they perform a Krylov projection each Newton iteration. For example, as discussed previously, Exp4 requires three Krylov projections per time step. If the Newton iteration converges quickly, e.g. if it converges in fewer than three iterations, it may require fewer Krylov projections per time step than Exp4. On the other hand, if the Newton iteration converges slowly, it may require more Krylov projections per time step. It is expected that as the size and stiffness of a problem grows, the Newton iteration will require more iterations to converge, and this may put Newton-Krylov methods at a disadvantage relative to exponential methods. Even if the Newton iteration converges quickly, if each of these iterations requires significantly more Krylov vectors than iterations of an exponential method, the latter can have better computational efficiency per time step than implicit Newton-Krylov methods.
3. Setup of numerical experiments

We are arguing that exponential methods are expected to outperform implicit methods due to a faster rate of convergence in the Krylov iteration for the type of matrix function they use. To test this idea we have implemented in MATLAB and compared the performance of several exponential, implicit and an explicit integrator on a set of stiff problems. In this section we describe the experimental setup, the integrators and the problems. The results of the numerical experiments are presented in Section 4.

3.1. Integrators

We compared the exponential integrators Exp4 (11), EpiRK4 (12), and ERow4 (13) with a Newton-Krylov implementation of the BDF4 implicit multistep method, two types of implicit Runge-Kutta methods and the explicit Runge-Kutta fourth order method. One of the implicit Runge-Kutta methods is a Rosenbrock method, which are considered to be particularly efficient for stiff problems [2], and the other is the Radau5 method. Below we compare and contrast the features of each method that impact their performance. Note that the features of the exponential methods were already discussed in the previous section.

Since our goal is to compare overall efficiency of the methods particularly from the perspective of Krylov-based implementations, we studied all the methods with constant time stepping to ensure an even comparison and to obtain a clear picture of the advantages and disadvantages of each integrator. Further, the Krylov iterations were run to a fixed error tolerance which was the same across all integrators and chosen to ensure the Krylov iterations did not limit the accuracy of the methods. While this can somewhat overcompute the Krylov iteration compared to an adaptive implementation, it was done so as to maintain a fair comparison of the number of Krylov iterations needed by each method.

For all problems the Jacobians were computed explicitly. Matrix-free calculations yield similar results but would have hidden the CPU costs inside the Krylov iterations which would have hampered conducting the cost breakdown.

For the exponential integrators, computation of the ϕ-functions of $H_m$ were computed using the Padé approximation algorithm of Higham [18].

The integrators were compared by picking an initial step size common to all the integrators and successively halving the step size over five sets of computations. The starting step size was $h = 0.01$ for all problems except the Allen-Cahn problem where it was chosen to be $h = 0.02$. A reference solution was computed using MATLAB’s ode15s integrator with absolute and relative tolerances set to $10^{-14}$ and the error was defined as the 2-norm of the difference between the computed solution and this approximation.

$BDF4$: The fourth order BDF scheme

$$y_{n+1} = \frac{12}{25} h F(t_{n+1}, y_{n+1} + \frac{48}{25} y_n - \frac{36}{25} y_{n-1} + \frac{48}{75} y_{n-2} - \frac{3}{25} y_{n-3})$$
is commonly used in modern codes to solve stiff problems [19], and is typically chosen over Adams-Moulton methods due to its superior stability properties. Each Newton iteration it must compute a Krylov projection using the matrix \((I - \frac{12}{25} hJ)\). In our comparisons, the three starting values (in addition to the initial value at \(t = 0\)) were computed using MATLAB’s ode15s integrator with absolute and relative tolerances set to \(10^{-14}\). In our performance comparisons below, this gives the first three starting values for BDF4 for free.

**Radau5:** Radau5 is a popular fifth-order implicit Runge-Kutta scheme which solves the following system at each time step

\[
\begin{pmatrix}
z_1 \\
z_2 \\
z_3 \\
y_{n+1}
\end{pmatrix} = A \begin{pmatrix}
hF(t_n + c_1 h, y_n + z_1) \\
hF(t_n + c_2 h, y_n + z_2) \\
hF(t_n + c_3 h, y_n + z_3)
\end{pmatrix},
\]

where

\[
A = \begin{pmatrix}
\frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{360} & -\frac{2+3\sqrt{6}}{225} \\
\frac{88+7\sqrt{6}}{1800} & \frac{296+169\sqrt{6}}{1800} & -\frac{2-3\sqrt{6}}{225} \\
\frac{1}{16+\sqrt{6}} & \frac{1}{16-\sqrt{6}} & \frac{1}{5}
\end{pmatrix},
\]

\[
c = \begin{pmatrix}
\frac{4-\sqrt{6}}{10} \\
\frac{4+\sqrt{6}}{10} \\
\frac{1}{10}\n\end{pmatrix}.
\]

To reduce computational cost an inexact Jacobian where all components are evaluated at \((t_n, x_n)\) is used and the \(3N \times 3N\) system within the Newton iteration is transformed into two \(N \times N\) systems, one real and one complex [2]. This contrasts with BDF4 which must solve a single real linear system of size \(N \times N\) each iteration. Complex number floating point multiplications are four times as expensive as real number multiplications, so solving the complex linear system is more expensive than solving the real system. As a result of these features Radau5 is more computationally expensive per time step compared to BDF4.

**Implicit fourth-order Rosenbrock method Ros4:** Rosenbrock methods are implicit Runge-Kutta methods designed to mitigate the need for solving the large \(3N \times 3N\) systems of regular implicit Runge-Kutta methods such as Radau5 by decoupling the stages [2]. The general form of a four stage Rosenbrock method is

\[
(I - h\gamma J)(k_i + \sum_{j=1}^{i-1} \frac{\gamma_{ij}}{\gamma} k_j) = F(y_{n+1}^{(i)}) + \sum_{j=1}^{i-1} \frac{\gamma_{ij}}{\gamma} k_j, \quad i = 1, \ldots, 4,
\]

\[
y_{n+1}^{(i)} = y_n + h \sum_{j=1}^{i-1} a_{ij} k_j
\]

\[
y_{n+1} = y_n + h \sum_{i=1}^{4} b_i k_i.
\]
Because of its structure this method does not need to resort to the Newton iteration, and instead at each stage a linear system of size $N \times N$ is solved. We use the GRK4T form of Rosenbrock method, so each time step requires four Krylov projections. The main difference with exponential methods is that those projections are used to compute a matrix rational instead of a matrix exponential.

\textit{Explicit fourth-order Runge-Kutta method RK4:} For an explicit method we used the classical fourth-order Runge-Kutta method given by

\begin{align*}
k_1 &= F(t_n, y_n), \\
k_2 &= F(t_n + \frac{1}{2} h, y_n + \frac{1}{2} k_1), \\
k_3 &= F(t_n + \frac{1}{2} h, y_n + \frac{1}{2} k_2), \\
k_4 &= F(t_n + h, y_n + h k_3), \\
y_{n+1} &= y_n + \frac{1}{6} h (k_1 + 2 k_2 + 2 k_3 + k_4).
\end{align*}

The method does not compute the Jacobian and makes no use of the Krylov iteration, so its primary computational cost is the four function evaluations. As such, its per-time-step cost is much lower than that of the Krylov-based methods. However, it is expected that the stability restrictions on the time step size will make this method uncompetitive with the other integrators if the problem is sufficiently stiff.

\subsection*{3.2. Test problems}

The following problems were used to compare the integrators. The outcome of the problems, each at two different sizes, are displayed in the precision diagrams of Fig. 1 and 2. The Allen-Cahn problem was additionally computed at a third smaller size of $N = 25^2$ to demonstrate how the stability of RK4 scales with $N$ (see Sec. 4).

In all the problems the $\nabla^2$ term was discretized using the standard second-order finite differences.

\textit{Allen-Cahn 2D.} Two-dimensional Allen-Cahn equation [20]:

\begin{align*}
  u_t &= \alpha \nabla^2 u + u - u^3, \quad x, y \in [0, 1], t \in [0, 0.2]
\end{align*}

with $\alpha = 0.1$, using Neumann boundary conditions and initial conditions given by $u = 0.4 + 0.1(x + y) + 0.1 \sin(10x) \sin(20y)$.

\textit{Brusselator 2D.} Two-dimensional Brusselator problem [21, 2]:

\begin{align*}
  u_t &= 1 + uv^2 - 4u + \alpha \nabla^2 u, \quad x, y \in [0, 1], t \in [0, 0.1], \\
v_t &= 3u - u^2 v + \alpha \nabla^2 v,
\end{align*}
with $\alpha = 0.2$. We used Dirichlet boundary conditions with initial and boundary values given by

\[
    u = 1 + \sin(2\pi x) \sin(2\pi y),
\]
\[
    v = 3.
\]

**Burgers.** One-dimensional Burgers equation:

\[
    u_t + uu_x = \nu u_{xx}, \quad x \in [0, 1], t \in [0, 1]
\]

with $\nu = 0.03$ and with Dirichlet boundary conditions and initial and boundary values given by $u = (\sin(3\pi x))^3(1 - x)^{3/2}$. The $uu_x$ term was discretized as

\[
    uu_x = \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x}, \quad i = 1, \ldots, N
\]

where $N$ is the number of spatial grid points chosen for the problem.

**Gray-Scott 2D.** Two-dimensional Gray-Scott problem [22]:

\[
    u_t = d_u \nabla^2 u - uv^2 + a(1 - u), \quad x, y \in [0, 1], t \in [0, 0.1],
\]
\[
    v_t = d_v \nabla^2 v + uv^2 - (a + b)v,
\]

with $d_u = 0.2$, $d_v = 0.1$, $a = 0.04$, and $b = 0.06$. Periodic boundary conditions were used and the initial conditions were given by

\[
    u = 1 - e^{-150(x - \frac{1}{2})^2 + (y - \frac{1}{2})^2},
\]
\[
    v = e^{-150(x - \frac{1}{2})^2 + 2(y - \frac{1}{2})^2}.
\]

**ADR 2D.** Two-dimensional advection-diffusion-reaction equation [11]:

\[
    u_t = \epsilon (u_{xx} + u_{yy}) - \alpha (u_x + u_y) + \gamma u(u - \frac{1}{2})(1 - u), \quad x, y \in [0, 1], t \in [0, 0.1],
\]

where $\epsilon = 1/100$, $\alpha = -10$, and $\gamma = 100$. Homogeneous Neumann boundary conditions were used and the initial conditions were given by $u = 256(xy(1-x)(1-y))^2 + 0.3$.

**Degenerate Nonlinear Diffusion 1D.** The degenerate nonlinear diffusion problem [23]:

\[
    \frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left[ u \frac{\partial u}{\partial x} \right] + u(1 - u),
\]

on the domain $-23 < x < 50$ with Dirichlet boundary conditions $u(-23, t) = 1$ and $u(50, t) = 0$, and with initial conditions

\[
    u(x, 0) = \begin{cases} 
        1 & \text{if } x < 0 \\
        e^{-1.3x} & \text{if } x > 0.
    \end{cases}
\]
4. Numerical Results

In this section we describe the results of our comparisons of exponential integrators with the explicit and implicit methods. We begin by using precision diagrams to show that exponential methods compete very well with the other methods, and then spend the remaining subsections breaking down the underlying reasons for their performance advantage. Our results show that faster Krylov convergence is in fact the primary advantage of exponential methods over the Krylov-based implicit techniques.

It can be seen from the precision diagrams (Fig. 1 & 2) that the exponential integrators generally outperformed the implicit methods. The performance curves of the exponential methods are generally well to the left of the curves for the implicit integrators, which means they used less CPU time to achieve the same level of accuracy. A minor exception was the degenerate nonlinear problem of the smallest size $N = 500$ for several small tolerance values. The data presented below demonstrates that the exception was due to the fact that for these parameters the nonlinear diffusion problem is simply not sufficiently stiff, and as the problem size was increased the performance of the exponential integrators superseded the implicit methods.

The three exponential integrators performed within 35% efficiency of each other. Since they were closer in performance to each other than to implicit methods, to analyze the results we first compare implicit integrators with Exp4, which is arguably to date the most well-known exponential integrator for general nonlinear problems. Then we will compare exponential integrators with respect to each other.

Before discussing comparative performance we note that for some problems Radau5 had to employ Householder orthogonalization to compute the Krylov projections instead of the modified Gram-Schmidt orthogonalization which was used in all other cases. The graphs in the precision diagrams where the Householder algorithm was used are labeled "Radau5-H". These are the cases where the modified Gram-Schmidt process suffers from roundoff error problems \[24, 25\] and the Krylov iteration breaks down due to the loss of orthogonality. Householder orthogonalization is quite robust and ensures that the Krylov iterations complete successfully in all cases. However, Householder orthogonalization is computationally more expensive than the modified Gram-Schmidt, which puts Householder-based integrators at a performance disadvantage. The exponential methods did not exhibit the same problems with the modified Gram-Schmidt algorithm since they required smaller basis sizes. Note however that very large scale applications could give rise to problems where even exponential integrators require a large basis size and consequently must employ the Householder algorithm.

Comparison with Ros4: It is simplest to compare exponential integrators with each implicit method individually and we begin with Ros4. Since here we are studying constant time step versions of the methods, to compare the relative performance of the integrators we fix the tolerance for the solution at a particular value and determine CPU time by interpolating along the precision diagram.
curves. Overall, except for the small size of the nonlinear diffusion problem, for all of the problems at most tolerances Ros4 required at least twice the CPU time of Exp4. Table 1 lists the tolerances for which the performance ratio between Ros4 and Exp4 was smallest and largest. The magnitude of the performance gap depended on the problem structure, size and the chosen tolerance and ranged from 32% to 547% improvement in the performance for Exp4 compared to Ros4. We also observe that the performance gap increased with problem size. Notably, Exp4 became more efficient than Ros4 at all tolerances for the large size of the nonlinear diffusion problem, eliminating the disadvantage it had at small tolerances on the small size of the problem. For the other problems, the increase varied greatly from as little as 1% as in the case of the Gray-Scott problem where the maximal gap increased from 274% to 275%, to as much as 203% as in the Brusselator problem where the minimal gap increased from 239% to 442%.

Ros4’s performance disadvantage with respect to the exponential integrators stems from its structure. It requires four Krylov projections per time step compared to three for the exponential methods. Those projections are also more expensive, since it uses rational functions instead of $\varphi$-functions. For small problems sizes and for small step sizes the performance gap with the exponential methods is modest. For large problem sizes and large step sizes, the performance gap is increased, e.g. the Brusselator problem with $N = 150^2$ where Ros4 was 547% of Exp4 at tolerance $5.0 \times 10^{-5}$.

Comparison with Radau5: Radau5 was the worst performing among the implicit integrators. In the best case of the small size of the degenerate nonlinear diffusion problem it required only 81% of the CPU time of Exp4 due to the problem placing low demands on the Krylov iterations. For all the other problems, at most tolerances it required well over five times the CPU time of Exp4. Furthermore, as problem size increased the performance gap with the exponential methods increased rapidly. For example, in the case of the Advection-Diffusion-Reaction problem, the gap widened by over 1000% when the problem was increased from size $N = 50^2$ to size $N = 150^2$. Note that some percentages in the table are marked with an asterisk. Those points were computed using the Householder algorithm, which is more expensive than the modified Gram-Schmidt. For four of the experiments, modified Gram-Schmidt was used for the small size of the problem and Householder for the large size of the problem, so it was natural that the gap widened due to the use of a more expensive algorithm for the large problem size. However, the Advection-Diffusion-Reaction problem used modified Gram-Schmidt for both problem sizes, and the gap still widened significantly, from 374% to 569% in the minimal case and from 3398% to 5550% in the maximal case. Householder was used for both sizes of the Burgers problem, and the gap increased by over 1.5 times in that case as well.

The poor performance of Radau5 is a consequence of the fact that it must compute two projections per Newton iteration, the second requiring expensive complex-number arithmetic operations, and that the Newton iteration converges more slowly compared to BDF4. In all cases it required at least two Newton iterations per time step and sometimes up to ten. This means it required computation of at least four Krylov projections per time step compared to three
### Table 1: Average Krylov vectors counts and total CPU time.

<table>
<thead>
<tr>
<th>(a) Relative performance between Ros4 and Exp4</th>
<th>Min. difference</th>
<th>Max. difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Error CPU time</td>
<td>Error CPU time</td>
</tr>
<tr>
<td></td>
<td>Exp4 Ros4 % Exp4</td>
<td>Exp4 Ros4 % Exp4</td>
</tr>
<tr>
<td>ADR:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>3.9e-06 0.75 1.27 169%</td>
<td>7.6e-02 0.26 0.75 292%</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>1.5e-06 2.83 7.13 252%</td>
<td>2.5e-03 3.18 15.30 482%</td>
</tr>
<tr>
<td>Allen-Cahn:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>1.5e-08 0.49 0.68 139%</td>
<td>6.7e-06 0.14 0.55 385%</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>1.4e-08 6.32 13.34 211%</td>
<td>8.2e-06 4.82 22.27 462%</td>
</tr>
<tr>
<td>Brusselator:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>4.5e-07 0.97 2.32 239%</td>
<td>2.6e-05 0.38 1.38 368%</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>4.2e-03 10.67 47.20 442%</td>
<td>5.0e-05 8.64 47.22 547%</td>
</tr>
<tr>
<td>Burgers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 500$</td>
<td>2.0e-11 4.62 7.70 167%</td>
<td>8.4e-08 2.51 7.82 311%</td>
</tr>
<tr>
<td>$N = 1500$</td>
<td>9.4e-10 12.89 29.27 227%</td>
<td>6.5e-08 16.82 59.47 353%</td>
</tr>
<tr>
<td>Gray-Scott:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>2.0e-07 0.47 0.85 183%</td>
<td>8.2e-06 0.29 0.79 274%</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>4.5e-08 8.46 18.07 214%</td>
<td>2.7e-05 9.19 25.25 275%</td>
</tr>
<tr>
<td>Degenerate Nonlinear Diffusion:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 500$</td>
<td>2.7e-09 32.15 22.53 70%</td>
<td>1.2e-05 3.86 8.54 221%</td>
</tr>
<tr>
<td>$N = 1500$</td>
<td>2.8e-08 88.88 117.39 132%</td>
<td>1.2e-04 22.70 68.00 300%</td>
</tr>
</tbody>
</table>

| (b) Relative performance between Radau5 and Exp4 | Min. difference | Max. difference |
|                                               | Error CPU time  | Error CPU time  |
|                                               | Exp4 Radau5 % Exp4| Exp4 Radau5 % Exp4|
| ADR:                                          |                |                |
| $N = 50^2$                                    | 4.7e-06 0.72 2.71 374% | 2.5e-02 0.25 8.34 3398% |
| $N = 150^2$                                   | 3.1e-09 4.85 27.60 569% | 2.5e-03 3.18 176.36 5550% |
| Allen-Cahn:                                   |                |                |
| $N = 50^2$                                    | 1.6e-10 0.74 1.53 208% | 4.3e-07 0.21 1.42 683% |
| $N = 150^2$                                   | 4.3e-09 6.77 111.63 1649%* | 1.4e-06 4.59 210.10 4582%* |
| Brusselator:                                  |                |                |
| $N = 50^2$                                    | 2.5e-07 1.01 2.00 197% | 1.2e-04 0.33 2.28 689% |
| $N = 150^2$                                   | 9.4e-06 10.69 225.57 2111%* | 4.6e-04 8.69 392.42 4516%* |
| Burgers:                                      |                |                |
| $N = 500$                                     | 3.7e-12 5.41 35.12 650%* | 5.3e-09 2.61 37.05 1419%* |
| $N = 1500$                                    | 4.1e-11 12.94 177.74 1373%* | 9.3e-09 14.52 336.94 2320%* |
| Gray-Scott:                                   |                |                |
| $N = 50^2$                                    | 9.2e-11 1.00 2.06 206% | 1.3e-06 0.38 2.22 583% |
| $N = 150^2$                                   | 4.2e-10 10.75 193.88 1804%* | 4.1e-06 8.32 262.32 3154%* |
| Degenerate Nonlinear Diffusion:               |                |                |
| $N = 500$                                     | 1.5e-09 33.70 27.31 81% | 1.3e-06 7.76 29.34 378% |
| $N = 1500$                                    | 1.3e-08 107.08 155.99 146% | 6.9e-06 27.10 400.27 1477% |

for the exponential methods. In addition, the basis size required for each of the projections was larger then that needed for any of the Krylov projections within an exponential integrator since Radau5 requires computation of rational rather then $\varphi$-functions. These two are the main reasons that Radau5 performed poorly compared to the exponential methods.
Comparison with BDF4: In most cases BDF4 was the best performing among the implicit methods. However, its overall performance was still worse compared to the exponential methods; for all problems all of the BDF4 performance curves lie to the right of exponential methods graphs on the precision diagrams. However, the same approach in comparing the performance gap that we used for Ros4 and Radau5 is misleading in this case. This is due to the fact that BDF4 appears to produce less accurate solutions for a given step size compared to exponential or other implicit methods. This causes the performance curves for BDF4 to be shifted up with respect to other graphs on the precision diagrams. Note that if a problem is stiff enough the performance curve tends to bend (e.g. both sizes of the the Burgers problem) so that lowering the step size actually decreases the CPU cost. This happens due to the fact that the complexity of Krylov iterations is not linear with basis size and we will discuss this point further in Section 4.0.2. In terms of comparing the two curves along a fixed tolerance line this means that we are comparing performance of BDF4 with a calculation where the CPU time of the exponential method is unnecessarily high, i.e. it is possible in this case to compute a more accurate solution with an exponential method for a lower computational cost for a smaller value of $h$. For example in the case of the Gray-Scott problem with $N = 150^2$ at tolerance $10^{-5}$, the CPU time for BDF4 solution is at its minimum but the time for Exp4’s solution is unnecessarily high. Lowering the tolerance would decrease the CPU time for Exp4 and still provide a more accurate solution. A clearer way to judge the size of the computational performance gap between the BDF4 and exponential methods is to compare them for a fixed size of $h$.

(a) Relative performance between BDF4 and Exp4

<table>
<thead>
<tr>
<th></th>
<th>Min. difference</th>
<th>Max. difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$h$</td>
<td>$h$</td>
</tr>
<tr>
<td></td>
<td>Exp4</td>
<td>BDF4</td>
</tr>
<tr>
<td>ADR:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>6.25e-04</td>
<td>0.86</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>6.25e-04</td>
<td>4.85</td>
</tr>
<tr>
<td>Allen-Cahn:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>1.25e-03</td>
<td>0.74</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>1.25e-03</td>
<td>6.77</td>
</tr>
<tr>
<td>Brusselator:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>1.25e-03</td>
<td>0.65</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>6.25e-04</td>
<td>10.69</td>
</tr>
<tr>
<td>Burgers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 500$</td>
<td>6.25e-04</td>
<td>5.41</td>
</tr>
<tr>
<td>$N = 1500$</td>
<td>6.25e-04</td>
<td>12.94</td>
</tr>
<tr>
<td>Gray-Scott:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 50^2$</td>
<td>1.25e-03</td>
<td>0.60</td>
</tr>
<tr>
<td>$N = 150^2$</td>
<td>1.25e-03</td>
<td>8.82</td>
</tr>
<tr>
<td>Degenerate Nonlinear Diffusion:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 500$</td>
<td>6.25e-03</td>
<td>34.66</td>
</tr>
<tr>
<td>$N = 1500$</td>
<td>6.25e-03</td>
<td>107.08</td>
</tr>
</tbody>
</table>

Table 2: Average Krylov vectors counts and total CPU time.
Table 2 lists data for the experiments where the minimal and maximal performance gap between BDF4 and Exp4 was exhibited for each problem when comparing at a fixed size of $h$. For fixed $h$ we see that the performance gap increased as the size of the problem grew in all cases. For some problems the increase was modest as is the case for the Gray-Scott problem where the performance gap increase from 116% to 136% for the minimal case and 154% to 202% for the maximal case. But for other problems the gap was more substantial, e.g., the Advection-Diffusion-Reaction problem where it grew from 108% to 170% in the minimal case and from 421% to 1063% in the maximal case.

Note that Ros4 and Radau5 have accuracy properties similar to exponential methods and the performance curves of exponential integrators are roughly at the same level. Thus either approach to comparing the performance gap (fixed tolerance or fixed time step size) yields a similar comparison between these implicit integrators and the exponential methods.

As opposed to other implicit methods, BDF4 usually required computation of fewer Krylov projections per time step than the exponential methods. It had to compute only one Krylov projection per Newton iteration, and for most problems it only required two Newton iterations per time step. Note however that for large scale problems we can expect the number of Newton iterations to grow, and BDF4 may need to compute an equal or greater number of Newton iterations (and thus Krylov projections) compared to the exponential methods. But even if BDF4 uses fewer Krylov projections, they are more expensive than those for the exponential methods to such a degree that in balance BDF4 performed worse than the exponential methods despite requiring fewer Krylov iterations. When problem size was increased, as with all implicit methods, the CPU cost per-projection increased more rapidly for BDF4 than for the exponential methods, and this is reflected in the performance gap increases seen in Table 2.

Comparison with RK4: Our results also confirm that exponential methods are expected to outperform explicit methods for problems which are sufficiently stiff. For highly stiff problems, RK4 performed significantly poorer than the other methods. For example, for the Burgers problem with $N = 1500$, to maintain stability RK4 took such small steps that it required more CPU time than all the other integrators, for all tolerances tested. For moderately stiff problems, RK4 was competitive for small problem size but began to fair worse as the problem size was increased. For example, for the Allen-Cahn problem with $N = 25^2$, to stably compute a solution RK4 required at least 39 time steps, for which it took 0.028 seconds of CPU time. The solutions of the exponential and implicit methods were computed using between 10 and 160 time steps. The per-step CPU cost of RK4 is much lower than the other methods, and regardless of how few time steps they used the exponential and implicit integrators always required more than 0.028 seconds of CPU time. As such, RK4 was more efficient than the other methods for that problem size. However, when the size was increased to $N = 150^2$ RK4 required at least 1310 steps, for which it took 20.9 seconds of CPU time. The exponential methods and BDF4 required fewer than 20.9 seconds regardless of how many steps they took. The maximum time
required by the exponential methods was 6.77 seconds for Exp4 to compute 160 steps. BDF4 required a maximum of 14.05 seconds for 10 steps, but only required 8.32 seconds for 80 steps. The higher cost for fewer steps is because the complexity of the Krylov iterations scales superlinearly with basis size (which will be discussed in later sections). Ros4 required less time for all but the two coarsest step sizes, requiring a maximum of 29.33 seconds for 10 steps, but less than 20.9 when computing between 40 and 160 steps. Radau5 remained more expensive than RK4 for all step sizes. Since they used much fewer time steps, the other methods naturally had less accuracy than RK4 with 1310 steps, but it was impossible to stably compute a solution with a smaller tolerance with RK4. These results provide a quantitative illustration of the well known fact that stability constraints make explicit integrators less efficient than more stable methods on sufficiently stiff problems [1, 2].

Comparative performance of the exponential integrators: While the exponential integrators performed similarly as a group in comparison to the implicit methods, there are still some notable aspects about their performance relative to each other. Compared to EpiRK4 and ERow4 which use faster converging higher-order $\varphi$-functions, Exp4 uses the slower converging $\varphi_1$ function for all three of its projections making its second projection more expensive. However, Exp4’s scaling of the Jacobian by $1/3$ makes its third projection require fewer Krylov vectors than the other two methods. How the efficiency of Exp4 compared with the other exponential methods was a matter of balance between these factors. On most problems Exp4 required less CPU time than the other two methods, particularly for large problems and at large step sizes where the Krylov iterations were most expensive. For example, for the large size of the Advection-Diffusion-Reaction problem, EpiRK4 is 29% more expensive and ERow4 is 21% more expensive than Exp4. In contrast, even for the large size of the Burgers problem Exp4 performed similarly to the other two methods. For that problem the Krylov basis sizes reduced rapidly with each successive projection so Exp4’s advantage on the third projection was not as important, balancing out with its lower efficiency on the second projection to give similar CPU time as the other methods. EpiRK4 and ERow4 had nearly identical CPU cost. For the same step size, the CPU times were always within 10% of each other, regardless of problem or step size.

4.0.1. Analysis of comparative performance as a result of Krylov iteration efficiency

In previous sections we saw that the exponential integrators performed better than implicit and explicit integrators. We also argued that the reason for the performance advantage is the reduced cost of the Krylov projections for the methods. In this section we present results supporting this claim.

First, we want to verify that Krylov projections in fact constitute the major portion of the cost in all of the algorithms. We used the profiler to measure the computational cost of the important portions of the methods, i.e. (i) the Krylov iterations, (ii) evaluation of the Jacobian $J$, (iii) calculation of the $\varphi$-functions of $H_m$, and (iv) the right-hand-side function $F$ evaluations. For almost
all the computations (i.e. 13 problems × 5 step sizes × 6 integrators) Krylov projections constituted the largest portion of the computational cost compared to the calculations of (ii)-(iv). There were two exceptions to this rule.

The first is the degenerate nonlinear diffusion problem at small step sizes for which the Krylov iterations accounted for only a minor fraction of the CPU time, leaving the Jacobian computations as the greatest expense. The low cost of the Krylov projections for this problem accounts for why the exponential integrators did not outperform the implicit integrators at small step size, particularly for the small problem size where the Krylov iterations had the lowest cost. For the large step sizes, particularly for the large problem size, the Krylov projections were the largest cost and the exponential methods had a sizable performance advantage.

The second exception is BDF4 for the smallest step sizes for the Allen-Cahn and Gray-Scott problems where the cost of computing the Krylov projections fell slightly below the next highest cost. For the remaining cases, the percentage of the total CPU time spent executing Krylov iterations ranged from 73% to 99.97% for large step sizes and was reduced to the range 37% to 88% for small step sizes, but even for small step sizes it remained larger then the next closest cost which was evaluation of the Jacobian or the right-hand-side function evaluations. Thus the efficiency of the Krylov projections portion of the algorithm had the largest effect on the overall cost of the method. For each integrator the total Krylov performance consisted of how many Krylov projections had been executed and how many Krylov vectors each of those projections required. In the following sections we demonstrate how those two aspects affect the performance of the methods.

Cost via number of Krylov vectors: Let us first look at the number of the Krylov vectors. This cost can be viewed from two perspectives: we can consider the total number of Krylov vectors taken each time step (i.e. sum the number of Krylov vectors taken by each of the projections in the method) or the average number of Krylov vectors per projection (Tables 3 and 4). For Ros4 and Radau5 both of these measures yield the same results. Both integrators compute more Krylov vectors per projection and also a larger total number of Krylov vectors than exponential methods. The gap in the number of vectors, both total and per-projection, is largest for coarse step sizes and is somewhat reduced for smaller size of $h$, but is never zero. The gap in the number of Krylov vectors between these implicit methods and exponential integrators grows as the stiffness of the problem is increased.

We illustrate these effects quantitatively with the Allen-Cahn problem which exhibited a typical outcome among the problems in the test suite. Some Krylov statistics for the problem are listed in Table 3. Using Exp4 as a representative of the exponential methods, we see that Ros4 and Radau5 always computed more Krylov vectors than Exp4 per projection for all step sizes, e.g. 37.1 vectors for Ros4 and 42 for Radau5 compared to just 22.7 for Exp4 for the first projection (first column) at coarse step size. The total number of computed vectors was higher as well, e.g. 1452 for Ros4 and 1836 for Radau5 compared to only 493 for Exp4 at coarse step size. The gap sizes shrank as the step size was
reduced but remained significant. For the smallest $h$, Ros4 computed about 10.8 vectors for all three projections. Exp4 computed 9.8 for the first, which was only marginally smaller, but 6.3 and 4.0 for the remaining two projections. In the first two Newton iterations, Radau5 computed two projections with 15 vectors and two with 10, which were both higher than the sizes for all three projections of Exp4. The gap in the total number of Krylov vectors was also reduced but remained significant with Ros4 computing 1723 vectors and Radau5 2413 vectors compared to only 804 for Exp4.

This example highlights an important structural difference between the methods. For the exponential methods in the products of type $f(A)b$ that have to be calculated the $b$ vectors used after the first projection are equal to the nonlinear remainder terms $R(Y)$ which have smaller magnitudes than the $b$ vectors for the first stage (which is the right-hand-side function $F$), causing the basis sizes for the second two projections to be smaller than that of the first. The $b$ vectors of Ros4 are not remainder terms but rather combinations of $F(y)$ and stage values $k_i$, which are not necessarily expected to decrease in magnitude. As such there is no falloff in basis size, so the gap with the exponential methods is even larger for the later projections. The basis sizes for Radau5 fall off as the error in the Newton iteration is reduced, but in all the problems the basis sizes for the first two Newton iterations, i.e. the first four projections, were larger than for all three of the exponential methods.

As with all the problems, for Allen-Cahn there was an increase in the difference in both the size of the basis and total vectors computed by the implicit methods compared to the exponential integrators, though in Table 1 we saw that the change in CPU time was modest for this problem so we expect the change in the vector count to be modest as well. For the coarsest step size, the ratio of the number of Krylov vectors for the first projection of Ros4 versus Exp4 increased from 1.63 to 1.81 in going from the small to large problem size. The other projections were similar. The ratio of the total number of Krylov vectors increased from 2.95 to 3.20. The inflation in the CPU cost was from 436% of Exp4 to 546%, which was larger than might be expected for the changes in vector count but there are two reasons for this. The first is that the cost of computing the Krylov vectors grows quadratically with the number of Krylov basis vectors $m$. (Specifically its $2m^2 N$ when using modified Gram-Schmidt orthogonalization, and $4m^2 N - 4/3m^3$ when using Householder orthogonalization [24]). Even a modest inflation in the extra number of vectors computed by an implicit method will result in a substantial increase in the CPU time. The second is that the larger basis sizes of the bigger problem cause the Krylov iterations to take up a greater proportion of the total CPU time, which causes the higher Krylov costs of the implicit methods to matter more. Similar increases in vector count happened for Radau5 though the CPU time went up more severely due to the use of the Householder orthogonalization for the larger problem. The gap in the number of Krylov vectors increased at smaller step sizes as well commensurate with the inflation in CPU time.

Cost via number of Krylov projections: The integrators require computation of different numbers of Krylov projections per step and the difference in total
CPU cost is a balance between the number of projections per step versus the number of vectors taken per projection. Ros4 always computes four projections per step and Radau5 required at least two Newton iterations (hence four projections) in our experiments. Thus, both methods required more projections per step and more vectors per projection than the exponential methods three projections resulting in higher CPU cost. Recall however that BDF4 usually required only two projections per step yet still had the higher CPU cost. From Table 3 we can see the reason for this is the the higher number of vectors taken per projection outweighs the smaller number of projections. For example in the case of the small size of the Allen-Cahn problem at the coarsest time step BDF4 required 51.7 and 31.3 vectors for the first and second projections whereas Exp4 required only 22.7 and 17.6 for the first two, but also required 9.0 vectors in a third projection. This balances out to the total number of vectors being similar, 590 total vectors for BDF4 compared to 490 for Exp4, but BDF4 required 221% the CPU time of Exp4. The disproportionate increase in CPU time comes from the quadratic scaling of cost with basis size $m$. BDF4 computes a total number of vectors similar to the exponential methods, but those vectors are more expensive due to the larger basis sizes and this results in a higher overall cost.

Obviously in cases where BDF4 requires more than two Newton iterations the performance gap was even greater. For the Advection-Diffusion-Reaction problem, BDF4 took as many as four Newton iterations. Some statistics for the problem are displayed in table 4. As before, there is still a sizable difference in basis sizes per projection between BDF4 and the exponential methods, but now the total Krylov vectors is no longer similar so the difference in CPU time becomes even greater, e.g. on the small problem size at coarse time step BDF4 took 1580 total vectors compared to only 639 for Exp4 resulting in 415% greater CPU time for BDF4.

Comparison of Krylov performance between exponential integrators: As we saw in the previous section and as Tables 3 and 4 confirm Exp4 takes slightly more Krylov vectors on the second projection but fewer on the third projection resulting in generally better performance compared to ERow4 and EpiRK4. As an example, for the Allen-Cahn problem with $N = 150^2$ at the coarsest step size, Exp4 used 53.0 vectors for its second projection compared to 48.8 and 52.9 for EpiRK4 and ERow4 respectively. However, it needed only 25.3 vectors for the third projection compared to 44.9 and 40.9 for EpiRK4 and ERow4. In balance, Exp4 computed fewer total Krylov vectors to give 10% better overall CPU performance. Across all the problems, Exp4 computed up to 20% fewer total Krylov vectors compared to the other two methods and typically 10% less. Comparing EpiRK4 with ERow4 we found the performance of these two methods to be quite similar to each other with the total number of vectors always within 6% and typically within 2%.

4.0.2. Krylov adaptivity

As we saw in the precision diagrams, reducing the step size can sometimes reduce the cost of the Krylov iterations so dramatically that computing a solution with a smaller value for $h$ results in a lower CPU time despite a larger number
of steps being computed. A particularly visible example of this is BDF4 on the Burgers problem with \( N = 1500 \) where the slope of a portion of the performance curve is positive (Fig. 5.2(b)). In many cases there is a transition point at which the slope changes sign, as for BDF4 used on the Burgers problem with \( N = 500 \) where the slope becomes negative at tolerance values of about \( 10^{-7} \).

The reason lowering step size can lower CPU cost is the Krylov iteration’s quadratic scaling of cost with basis size. We can see in Tables 3 and 4 that the number of Krylov vectors needed per projection decreases by a factor of 1.5 to 2.0 each time \( h \) is halved (although it varies somewhat with problem and step size). Because of the quadratic cost scaling, each time the step size is halved the CPU time per projection is reduced between 1.5 and 2.0 times, i.e. by a factor larger than two. If the Krylov projections were the entire computational cost, halving the step size would always lower the CPU time. However, as the cost of the Krylov projections decrease they account for an ever smaller percentage of the total computational cost and other components, such as the calculation of the Jacobian, become more relevant. As a result, at some point lowering the step size further starts to increase the overall CPU time.

This crossover phenomenon is meaningful for how variable time step methods should be implemented. If lowering the step size reduces CPU cost, it is more cost efficient to compute with smaller \( h \) even if the extra accuracy is not needed for coarse tolerances. However if the step size is lowered too much, the CPU time will start to increase. This suggests the need for an adaptivity algorithm which is able to adjust the step size to find the “sweet spot” step size for which CPU time is lowest. Early attempts at developing such Krylov adaptivity algorithms can be found in [7, 26], but so far there is only limited study of how effectively these algorithms find an optimal step size.

### 5. Performance optimization of exponential integrators

In this section we demonstrate that the performance on exponential integrators can be further improved with careful design of a method. Recently a new class of EpiRK methods have been introduced [13]. The general form of EpiRK schemes is

\[
Y_i = y_n + a_{i1} \psi_{i1}(g_{i1} h J_n) h F_n + \sum_{j=2}^{i-1} a_{ij} \psi_{ij}(g_{ij} h J_n) h \Delta^{(j-1)} R(y_n), \quad i = 1, \ldots, (s-1)
\]

\[
y_{n+1} = y_n + b_1 \psi_{s1}(g_{s1} h J_n) h F_n + \sum_{j=2}^{s} b_j \psi_{sj}(g_{sj} h J_n) h \Delta^{(j-1)} R(y_n),
\]

where \( \psi_{ij}(z) \) functions are defined as

\[
\psi_{ij}(z) = \sum_{k=1}^{s} p_{ijk} \varphi_k(z),
\]

(15)

\( s \) is the number of stages and the divided-differences \( \Delta^{(j-1)} R(y_n) \) are computed using the nodes \( y_n, Y_1, Y_2, \ldots, Y_{s-1} \). The coefficients \( a_{ij}, g_{ij}, b_j \) and \( p_{ijk} \) are
chosen based on the order conditions. The flexibility in choosing the coefficients offered by this ansatz allows development of more efficient schemes. Specifically, it was shown in [13] that it is possible to derive a fifth-order EpiRK method EpiRK5S3 which has the same number of stages as the EpiRK4 scheme. The coefficients for the fifth order EpiRK5S3 method are listed in Table 5.

From comparing the structure of the EpiRK5S3 scheme with other exponential schemes discussed above, we can expect this method to be the most efficient integrator for two main reasons. First, just like the fourth-order exponential methods EpiRK4, ERow4 and Exp4, the EpiRK5S3 scheme requires only three Krylov projections per time step, but since the method is fifth-order it provides more accuracy. Second, since the coefficients $g_{ij}$ in (14) scale the Jacobian $J_n$ we can expect fewer Krylov vectors needed for calculation of the terms $\psi_{ij}(g_{ij}hJ)b$ if $g_{ij} < 1$. Inspecting the coefficients $g_{ij}$ in Table 5 for a fixed $j$ we can see that the second and the third Krylov projections have the maximum $g_{ij}$ ($j = 2, 3$) coefficients smaller than 1 and therefore we can expect that EpiRK5S3 will require fewer Krylov vectors for these projections compared to schemes where evaluations must be made with coefficients $g_{ij} \geq 1$. Figure 3 and Table 6 illustrate how these features of the scheme translate to CPU savings.

$$\begin{bmatrix} a_{11} \\ a_{21} & a_{22} \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.4165701558065186 \\ 0.8624674370127457 \\ 1.0 \end{bmatrix}$$

$$\begin{bmatrix} g_{11} \\ g_{21} & g_{22} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.4165701558065186 \\ 0.8624674370127457 \\ 1.0 \end{bmatrix}$$

Table 5: Coefficients of fifth order EpiRK methods.

Fig. 3 provides precision diagrams comparing the performance of EpiRK5S3 with the other exponential methods for advection-diffusion-reaction, Allen-Cahn, Brusselator and Gray-Scott problems. To accommodate the increased accuracy of EpiRK5S3, the integrators were run with smaller tolerances for the Krylov iterations and at larger step sizes than for the comparisons with the implicit integrators. The EpiRK5S3 method exhibits better performance than other exponential integrators. The statistics of the Krylov projections performance for the Gray-Scott problem in Table 6 illustrates that computational advantage of the scheme is due to more than just the higher order of the method. For brevity, the statistics of the other problems are left out but the results are similar. The Krylov basis size for the EpiRK5S3’s first projection is similar to the other methods as the Jacobian is unscaled except by $h$ for all the methods in this case. However, the basis sizes of the second and third projections are lower than for the other methods. For example, in the case of $h = 0.02$ EpiRK5S3’s second projection has a basis size of 76.4 vectors, whereas the basis size for the method with the next smallest basis, EpiRK4, is 87.2 vectors which gives
a savings of 10.8 vectors. Exp4 has the largest basis size of all the methods at 93.8 vectors, a 17.4 vector difference with EpiRK5S3. However, the Jacobian of EpiRK5S3’s second projection is scaled only by $g_{32} = 0.73$ times $h$, so we expect only a modest savings. Its third projection has the Jacobian scaled by $g_{33} = 0.33$ times the step size, so we expect the savings to be greater in that case. Looking again at the case when $h = 0.02$, the basis size for EpiRK5S3’s third projection is 45.0 vectors. The method with the next smallest basis size is Exp4 with a basis size of 46.8 vectors, which is a difference of 1.8 vectors. However, Exp4 also has the Jacobian of its third projection scaled by 0.33 times $h$ so the small difference is expected. It’s worth reiterating that Exp4 uses the $\varphi_1$ function for its third projection while EpiRK5S3 uses higher-order functions, accounting for its small advantage. The remaining methods do not scale the Jacobian beyond multiplying by $h$ so we expect them to have much poorer performance. Indeed, the next best method is ERow4 with a basis size of 78.0 vectors, a difference of 33 vectors compared to EpiRK5S3, i.e. 173% as many. Overall, these savings result in higher efficiency of EpiRK5S3. For $h = 0.02$, EpiRK5S3 requires only 84% of the CPU time of the next best method, Exp4. As the step size gets smaller, the performance advantage of EpiRK5S3 shrinks but remains non-trivial. In conclusion, we can see that scaling the Jacobian with favorable coefficients gives significant reduction in Krylov basis size resulting in better overall performance, making it an important design criteria when deriving new methods.

<table>
<thead>
<tr>
<th></th>
<th>Average # of Krylov vectors per step</th>
<th>Proj. 1</th>
<th>Proj. 2</th>
<th>Proj. 3</th>
<th>Proj. 4</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>Total CPU time</th>
<th>% Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = 0.02$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EpiRK5S3</td>
<td>106.4</td>
<td>76.4</td>
<td>45.0</td>
<td>n/a</td>
<td>3</td>
<td>1130</td>
<td>12.9</td>
<td>84%</td>
<td></td>
</tr>
<tr>
<td>EpiRK4</td>
<td>106.8</td>
<td>87.2</td>
<td>82.4</td>
<td>n/a</td>
<td>3</td>
<td>1382</td>
<td>16.6</td>
<td>109%</td>
<td></td>
</tr>
<tr>
<td>ERow4</td>
<td>103.8</td>
<td>93.4</td>
<td>78.0</td>
<td>n/a</td>
<td>3</td>
<td>1376</td>
<td>16.9</td>
<td>110%</td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>107.2</td>
<td>93.8</td>
<td>46.8</td>
<td>n/a</td>
<td>3</td>
<td>1239</td>
<td>15.3</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>$h = 0.01$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EpiRK5S3</td>
<td>63.5</td>
<td>45.9</td>
<td>26.8</td>
<td>n/a</td>
<td>3</td>
<td>1362</td>
<td>11.1</td>
<td>85%</td>
<td></td>
</tr>
<tr>
<td>EpiRK4</td>
<td>65.0</td>
<td>52.4</td>
<td>48.4</td>
<td>n/a</td>
<td>3</td>
<td>1653</td>
<td>13.7</td>
<td>105%</td>
<td></td>
</tr>
<tr>
<td>ERow4</td>
<td>63.1</td>
<td>56.9</td>
<td>45.3</td>
<td>n/a</td>
<td>3</td>
<td>1653</td>
<td>13.8</td>
<td>105%</td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>65.3</td>
<td>57.0</td>
<td>27.5</td>
<td>n/a</td>
<td>3</td>
<td>1498</td>
<td>13.1</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>$h = 0.001$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EpiRK5S3</td>
<td>39.4</td>
<td>27.2</td>
<td>15.8</td>
<td>n/a</td>
<td>3</td>
<td>1646</td>
<td>10.1</td>
<td>84%</td>
<td></td>
</tr>
<tr>
<td>EpiRK4</td>
<td>40.4</td>
<td>31.1</td>
<td>27.4</td>
<td>n/a</td>
<td>3</td>
<td>1978</td>
<td>12.6</td>
<td>104%</td>
<td></td>
</tr>
<tr>
<td>ERow4</td>
<td>39.3</td>
<td>34.2</td>
<td>24.8</td>
<td>n/a</td>
<td>3</td>
<td>1964</td>
<td>12.2</td>
<td>101%</td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>40.1</td>
<td>34.1</td>
<td>16.2</td>
<td>n/a</td>
<td>3</td>
<td>1809</td>
<td>12.1</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Average Krylov vectors counts and total CPU time.

6. Comparisons of variable time step implementations

In the previous sections we have presented a detailed analysis that illustrated how each part of an integrator effects its overall performance. In order to clearly
demonstrate how the structure of a method and the parts it is comprised of, effect the performance we needed to use the constant step sizes in our experiments. However, this raises the question of whether the computational savings predicted by these experiments are still available when these methods are used in the context of variable time step algorithms. In this section, we address this issue and validate our results using comparisons between a well-tested and widely available implementation of a variable step size implicit integrator and a new variable step size exponential method.

As a benchmark implicit integrator we choose the ROWMAP implementation [27] of the GRTK4T implicit Rosenbrock method (Ros4). The core Rosenbrock scheme of this code was also used above in the constant time step experiments. The ROWMAP method, however, is a variable time step implementation that was specifically created to reduce the computational cost of Krylov projections per step. This goal was accomplished by employing the MAP (multiple Arnoldi process) algorithm which reuses the Krylov basis of the first stage of Ros4 in subsequent stages by extending it by a fixed number of additional vectors. Specifically, the Krylov basis for the first stage is computed using the usual Arnoldi process with the basis size determined based on the specified tolerance. Rather than computing the basis of the second stage from scratch, the MAP algorithm reuses the basis of the first stage by supplementing it by three more Krylov vectors. Likewise, the third stage extends the basis of the second stage with an additional vector, and the fourth stage extends the basis of the third stage by three vectors. As a result, only seven more vectors are computed in addition to the basis of the first stage. It was shown in [28] that using MAP preserves the fourth order of the method. Here we use the MATLAB implementation of ROWMAP algorithm available at http://numerik.mathematik.uni-halle.de/forschung/software.

The variable step size exponential integrator we used is the fifth-order EpiRK5P1, a newly derived method from the class of EpiRK integrators described in section 5. This algorithm was implemented using the adaptive Krylov projection algorithm proposed by Niesen and Wright [26]. The detailed description of the adaptive EpiRK methods can be found in [29]. Here we outline the main ideas behind the structure of the method. The adaptive EpiRK methods employ the Niesen-Wright adaptive Krylov projection algorithm which replaces computation of one large, computationally expensive Krylov basis needed to evaluate a linear combination of \( \varphi \)-functions-vector products of the form
\[
u(t) = \varphi_0(tA)b_0 + \varphi_1(tA)b_1 + \varphi_2(tA)b_2 + \ldots + \varphi_p(tA)b_p, \quad A \in \mathbb{R}^{N \times N}, \quad b_i \in \mathbb{R}^N,
\]
(16)
at \( t = 1 \) with several cheaper Krylov projections to approximate \( u(t) \) with \( 0 < t < 1 \). It is based on the observation by Skaflestad and Wright [30] that \( u(t) \) is the solution to the ODE
\[
u'(t) = Au(t) + b_1 + tb_2 + \ldots + \frac{t^{p-1}}{(p-1)!}b_p, \quad u(0) = b_0,
\]
(17)
and if \( t_0 = 1 < t_1 < \ldots < t_k < t_k+1 < \ldots < t_{end} = 1 \) then values \( u(t_i) \) can be
computed iteratively using the exact formula

\[
 u(t_{k+1}) = \varphi_0(\tau_k A) u(t_k) + \sum_{i=1}^{p} \tau_k^i \varphi_i(\tau_k A) \sum_{j=1}^{p-i} \frac{\tau_k^j}{j!} b_{i+j}, \quad \tau_k = t_{k+1} - t_k. \tag{18}
\]

The recurrence relation \( \varphi_q(A) = \varphi_{q+1}(A) A + \frac{1}{q!} \) can be employed to express equation (18) in a simplified form

\[
 u(t_{k+1}) = \tau_k^p \varphi_p(\tau_k A) w_p + \sum_{j=0}^{p-1} \frac{\tau_k^j}{j!} w_j, \tag{19}
\]

with \( w_j \)'s computed recursively as

\[
 w_0 = u(t_k), \quad w_j = A w_{j-1} + \sum_{l=0}^{p-j} \frac{\tau_k^l}{l!} b_{j+l}, \quad j = 1, \ldots, p. \tag{20}
\]

Linear combination (16) can then be adaptively computed by stepping equation (19) over a set of subintervals \( 0 = t_0 < t_1 < \ldots < t_k < t_{k+1} = t_k + \tau_k < \ldots < t_{end} = 1 \) and evaluating each term \( \varphi_p(\tau_k A) w_p \) using a Krylov projection. The computational tradeoff is that the series of Krylov projections for \( \varphi_p(\tau_k A) w_p \) for a scaled matrix \( \tau_k A \) require only a small Krylov basis compared to the basis size needed for evaluating \( \varphi_p(A) b_p \). Computing a series of such terms with small basis is found to be computationally cheaper than computing (16) with a single large Krylov basis given that the complexity of the Arnoldi iteration scales quadratically with the basis size. The values \( \tau_k \) in the algorithm are chosen adaptively using error estimates and the cost function, the details of this selection can be found in [26].

The coefficients of EpiRK5P1 are chosen so as to allow the method to use Niesen-Wright adaptivity while preserving the projection minimizing feature discussed in section 2, i.e. the terms \( f(A)b \) and \( f(cA)b \) sharing the same \( b \) vector can still be computed with the same Krylov basis. Interpreted through the general form of EpiRK methods (14), this projection minimizing property is equivalent to computing terms associated with coefficients \( g_{1j}, g_{2j}, \ldots, g_{sjj} \), with a single Krylov projection, for each \( j \). This feature can be retained in an algorithm with Niesen-Wright adaptivity since the terms \( \psi_{1j}(g_{1j}A)b, \psi_{2j}(g_{2j}A)b, \ldots, \psi_{sjj}(g_{sjj}A)b \) can be calculated using (19) with a single sweep of steps over subinterval \( 0 = t_0 < t_1 < \ldots < t_k < t_{k+1} = t_k + \tau_k < \ldots < t_{end} = 1 \) if the functions \( \psi_{1j}(z) \) consist of a single \( \varphi_k(z) \)-function, for some \( k \), and not a linear combination of \( \varphi_k(\tau_k(z))'s \). Such approach works as follows. Without loss of generality, let \( g_{1j} < g_{2j} < \ldots < g_{sjj} \) and \( \psi_{1j}(z) = \psi_{2j}(z) = \ldots = \psi_{sjj}(z) = \varphi_k(z) \) for some \( k \). Note that in case of a single \( \varphi_k(z) \), equation (16) reduces to \( u(t) = t^k \varphi_k(tA)b_k \). All of the terms \( \psi_{1j}(g_{1j}A)b, \psi_{2j}(g_{2j}A)b, \ldots, \psi_{sjj}(g_{sjj}A)b \) can then be computed in a single sweep over \( 0 = t_0 < t_1 < \ldots < t_k < t_{k+1} = t_k + \tau_k < \ldots < t_{end} = 1 \) by observing that \( \varphi_k(g_{jk}A)b \) is equal to computing the now reduced form of \( u(t) \) at time \( t = g_{jk}/g_{sk} \) and dividing by \( t^k \). EpiRK5P1 is constructed with
\[ \psi_{i1}(z) = \varphi_1(z), \psi_{i2}(z) = \varphi_1(z), \psi_{i3}(z) = \varphi_3(z). \] It is a fifth order method with coefficients listed in Table 7. It is worth noting that the flexibility of the general structure of the EpiRK class of methods allows the fifth order methods with only three stages to be constructed, while the accuracy of previously proposed exponential integrators with three stages did not exceed the fourth order. This is an advantage of the EpiRK class over both the exponential and implicit forms of the Rosenbrock methods. An embedded fourth order EpiRK method was also derived to provide the automatic step size control mechanism; the coefficients are the same as for EpiRK5P1 except that \( g_{32} = 0.5 \), and \( g_{33} = 1.0 \).

**Table 7: Coefficients of EpiRK5P1.**

The variable step adaptive EpiRK5P1 and ROWMAP methods were compared on the six problems from section 3.2 over the same time intervals. The comparisons were done twice - with two different choices of how the Jacobian was evaluated. In a first set of comparisons, EpiRK5P1 computed an explicit exact (i.e. not numerically differentiated) Jacobian matrix and computed matrix-vector products \( Jv \) using matrix multiplication, while ROWMAP used a matrix-free first-order finite differences estimate of terms \( Jv \). Just as in [27], ROWMAP was tested and found to be not significantly less accurate when using the numerical approximation compared to using an explicit Jacobian. In a second set of comparisons, EpiRK5P1 also used a finite differences approximation of terms \( Jv \). The first set of comparisons is discussed here and the second set in a later paragraph. In the first set, except for the degenerate nonlinear diffusion problem, both integrators were compared over the range of tolerances \( Atol = Rtol = \{10^{-2}, 10^{-3}, ..., 10^{-7}\} \). In the case of the degenerate nonlinear diffusion problem, the EpiRK5P1 method used tolerances \( Atol = Rtol = \{10^{-2}, 10^{-3}, ..., 10^{-8}\} \), while ROWMAP used tolerances scaled by a factor of \( 10^{-3} \) of those for EpiRK5P1 to make the performance of the methods more comparable (Fig. 4). The tolerances for the Krylov process were not kept fixed as in the constant time step case but rather chosen relative to the accuracy requirements of the current time step. Specifically, EpiRK5P1 stopped the Krylov process when \( res < 0.1 \ast h_n \ast \min(Atol, Rtol \ast ||y_n||) \), where \( res \) is the Krylov residual and \( 0.1 \) is a safety factor. ROWMAP terminated the Krylov process when \( h_n \ast \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( \frac{res}{Atol + Rtol + \text{abs}(y_n)} \right)^2 } < 0.1 \).

The results of the comparisons are shown in Fig. 4. It can be seen that
EpiRK5P1 generally outperforms ROWMAP, particularly for fine tolerances. While the use of Niesen-Wright adaptivity with the exponential integrator and the MAP algorithm with the implicit integrator makes direct comparison of Krylov performance difficult, the profiler shows that computing the Krylov projections remains as the dominant computational cost of the algorithms. To that extent the generally better performance of EpiRK5P1 can be attributed to an overall lower Krylov cost, consistent with what was seen in the constant time step case. In the case of the degenerate nonlinear diffusion problem, while EpiRK5P1 performed better for small tolerances, ROWMAP did better for coarse tolerances. As noted in section 4, this problem is not very stiff and the difference in performance between exponential integrators and implicit integrators is not significant. EpiRK5P1 performed better at fine tolerances due to its higher order.

Note that the ROWMAP algorithm and implementation have been developed and optimized over a relatively long time period [27], while the variable step EpiRK methods with Niesen-Wright adaptivity are a very recent development [29]. The resulting exponential algorithm can be further optimized and perfected. In particular, the error estimators and the cost functions within the Niesen-Wright adaptivity algorithm can be improved. An illustration of this is the precision diagram for the Burgers equation in Fig. 4(d). The performance of the EpiRK5P1 suffered at coarse tolerances because the Niesen-Wright adaptivity algorithm performed suboptimally when the Jacobian was scaled coarsely by a large step size $h$. As the tolerances tightened and the Jacobian was scaled better, the adaptivity algorithm made better choices of the substep sizes $\tau_k$. Our preliminary results show that the error estimators and the cost function used by the Niesen-Wright algorithm can be further refined and we will report on the improved adaptive EpiRK methods in future publications. The comparisons with inexact Jacobian presented in the next paragraph further illustrate this point, since the error estimators and the cost functions have not been adjusted to account for the inexact Jacobian approximation.

The second set of comparisons, where both integrators used a finite differences approximation of terms $Jv$, are shown in Fig. 5. Except for the case of the Burgers and the degenerate nonlinear diffusion problem, both integrators used tolerances $Atol = Rtol = \{10^{-2}, 10^{-3}, ..., 10^{-7}\}$. For better relative comparison, for the Burgers problem EpiRK5P1 used tolerances $Atol = Rtol = \{10^{-2}, 10^{-3}, ..., 10^{-7}\}$ while ROWMAP used tolerances $10^{-3}$ of those of EpiRK5P1, and on the degenerate nonlinear diffusion problem EpiRK5P1 used tolerances $Atol = Rtol = \{10^{-2}, 10^{-3}, ..., 10^{-11}\}$ while ROWMAP used tolerances $10^{-2}$ those of EpiRK5P1. Compared to the case when using an exact Jacobian, EpiRK5P1 suffered from reduced accuracy and poorer Krylov performance. When using an explicit Jacobian matrix, the Niesen-Wright algorithm uses sparsity information about the matrix in the cost function used to adaptively choose basis sizes and values of $\tau$. With the numerical estimate that information is unavailable and the current implementation falls back to a less accurate default estimate making the adaptivity perform less optimally. Compared to ROWMAP, EpiRK5P1 also exhibited more sensitivity to approx-
imation error in the Jacobian and loses overall accuracy when using a finite difference estimate. Nevertheless, despite the performance reduction, EpiRK5P1 generally compared well with ROWMAP.

In summary, much work remains to be done in optimizing the performance of adaptive exponential integrators, but early comparisons show that even newly developed adaptive exponential schemes exhibit promising performance compared to implicit integrators.

7. Conclusions and future work

In this paper we demonstrated that new exponential methods can perform better than some of the implicit methods typically used for large stiff problems. We have identified the reason for their performance advantage being the efficiency of the Krylov projections in evaluation of exponential-like matrix functions compared to rational matrix functions required by the implicit methods. These results represent one of the first careful numerical studies that provide a quantitative insight into what type of computational savings one might expect in using the latest exponential integrators compared to standard methods. The analysis details how the structure of an integrator, i.e. the number and the nature of Krylov projections it requires, affects its performance and provides guidelines for constructing and implementing efficient exponential integrators. These results instruct selection of appropriate time integrators for other problems. In addition, we have demonstrated computational efficiency of the newly introduced three-stage fifth order EpiRK methods. We have verified the performance advantages of the core EpiRK schemes when they are employed as constant time step integrators and when these methods are developed into variable step size adaptive integrators. The work has highlighted the importance of development of effective adaptive strategies and the promising research directions in this area. Larger scale problems and parallel implementations of the methods need to be studied and analyzed. We plan to address these questions in future publications.

8. Acknowledgements

This work was supported in part by a grant from the U.S. Department of Energy, Office of Science, Offices of Advanced Scientific Computing Research, and Biological & Environmental Research through the U.C. Merced Center for Computational Biology #DE-FG02-04ER25625. The authors would also like to thank Will Wright and Rudiger Weiner for helpful discussions.


Figure 1: Precision diagrams for the Advection-Diffusion-Reaction, Allen-Cahn, and Brusselator problems. Note that the axes scale changes from graph to graph.
Figure 2: Precision diagrams for the Burgers, Gray-Scott and Degenerate Nonlinear Diffusion problems. Note that the axes scale changes from graph to graph.
Figure 3: Precision diagrams comparing the coefficient-optimized EPIRK5S3 method to the other exponential methods.
Figure 4: Precision diagrams comparing variable time step implementations of Krylov-adaptive EpiRK5-P1 with ROWMAP-GRK4T. EpiRK5P1 uses an exact Jacobian while ROWMAP uses a finite differences approximation of the Jacobian.
Figure 5: Precision diagrams comparing variable time step implementations of Krylov-adaptive EpiRK5-P1 with ROWMAP-GRK4T. Both methods use a finite differences approximation of the Jacobian.
### Table 3: Average Krylov vectors counts and total CPU time. Note: BDF4 and Radau5 list the projections of their first four Newton iterations in columns Proj. 1, Proj. 2, etc.

#### (a) 2D Allen-Cahn problem with $N = 50^2$

<table>
<thead>
<tr>
<th></th>
<th>Average # of Krylov vectors per step</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proj. 1</td>
<td>Proj. 2</td>
<td>Proj. 3</td>
<td>Proj. 4</td>
</tr>
<tr>
<td></td>
<td>h = 0.02:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>22.7</td>
<td>17.6</td>
<td>9.0</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>22.4</td>
<td>16.0</td>
<td>14.2</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>22.2</td>
<td>17.6</td>
<td>13.5</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>51.7</td>
<td>31.3</td>
<td>1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>37.1</td>
<td>36.6</td>
<td>35.9</td>
<td>35.6</td>
</tr>
<tr>
<td>Radau5</td>
<td>42</td>
<td>44</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>h = 0.01:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>14.7</td>
<td>10.5</td>
<td>5.5</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>14.6</td>
<td>9.3</td>
<td>7.5</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>14.3</td>
<td>10.5</td>
<td>6.9</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>32.6</td>
<td>16.7</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>20.2</td>
<td>20.0</td>
<td>19.6</td>
<td>19.3</td>
</tr>
<tr>
<td>Radau5</td>
<td>25</td>
<td>24</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>h = 0.005:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>9.8</td>
<td>6.3</td>
<td>4.0</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>9.9</td>
<td>5.6</td>
<td>4.3</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>9.8</td>
<td>6.3</td>
<td>4.2</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>18.7</td>
<td>8.4</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>10.8</td>
<td>10.9</td>
<td>10.8</td>
<td>10.7</td>
</tr>
<tr>
<td>Radau5</td>
<td>15</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

#### (b) 2D Allen-Cahn problem with $N = 150^2$

<table>
<thead>
<tr>
<th></th>
<th>Average # of Krylov vectors per step</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proj. 1</td>
<td>Proj. 2</td>
<td>Proj. 3</td>
<td>Proj. 4</td>
</tr>
<tr>
<td></td>
<td>h = 0.02:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>64.7</td>
<td>53.0</td>
<td>25.3</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>64.8</td>
<td>48.8</td>
<td>44.9</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>62.7</td>
<td>52.9</td>
<td>40.9</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>159.7</td>
<td>97.9</td>
<td>7.86</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>117.0</td>
<td>115.0</td>
<td>113.8</td>
<td>112.5</td>
</tr>
<tr>
<td>Radau5-H</td>
<td>129</td>
<td>131</td>
<td>87</td>
<td>89</td>
</tr>
<tr>
<td>h = 0.01:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>39.1</td>
<td>31.1</td>
<td>14.5</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>40.0</td>
<td>28.2</td>
<td>23.2</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>38.6</td>
<td>30.9</td>
<td>20.9</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>98.2</td>
<td>52.1</td>
<td>1.0</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>67.2</td>
<td>65.8</td>
<td>65.2</td>
<td>64.6</td>
</tr>
<tr>
<td>Radau5-H</td>
<td>73</td>
<td>73</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>h = 0.005:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp4</td>
<td>24.7</td>
<td>18.1</td>
<td>8.2</td>
<td>n/a</td>
</tr>
<tr>
<td>Epi RK4</td>
<td>24.9</td>
<td>16.0</td>
<td>11.8</td>
<td>n/a</td>
</tr>
<tr>
<td>ERow4</td>
<td>24.4</td>
<td>17.9</td>
<td>9.4</td>
<td>n/a</td>
</tr>
<tr>
<td>BDF4</td>
<td>54.4</td>
<td>25.2</td>
<td>1.0</td>
<td>n/a</td>
</tr>
<tr>
<td>Ros4</td>
<td>34.3</td>
<td>33.5</td>
<td>33.6</td>
<td>33.2</td>
</tr>
<tr>
<td>Radau5-H</td>
<td>43</td>
<td>43</td>
<td>31</td>
<td>30</td>
</tr>
</tbody>
</table>

#### (b) 2D Allen-Cahn problem with $N = 150^2$
(a) 2D Advection-Diffusion-Reaction problem with $N = 50^2$

<table>
<thead>
<tr>
<th></th>
<th>Proj. 1</th>
<th>Proj. 2</th>
<th>Proj. 3</th>
<th>Proj. 4</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>Total CPU time</th>
<th>% Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = 0.01$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{Exp4}$</td>
<td>25.9</td>
<td>24.3</td>
<td>13.7</td>
<td>n/a</td>
<td>3</td>
<td>639</td>
<td>0.26</td>
<td>100%</td>
</tr>
<tr>
<td>$\text{EpiRK4}$</td>
<td>25.9</td>
<td>22.7</td>
<td>22.2</td>
<td>n/a</td>
<td>3</td>
<td>708</td>
<td>0.25</td>
<td>96%</td>
</tr>
<tr>
<td>$\text{ERow4}$</td>
<td>25.2</td>
<td>23.9</td>
<td>20.3</td>
<td>n/a</td>
<td>3</td>
<td>694</td>
<td>0.26</td>
<td>100%</td>
</tr>
<tr>
<td>$\text{BDF4}$</td>
<td>73.4</td>
<td>65.9</td>
<td>54.6</td>
<td>31.9</td>
<td>4</td>
<td>1580</td>
<td>1.08</td>
<td>415%</td>
</tr>
<tr>
<td>$\text{Ros4}$</td>
<td>42.8</td>
<td>42.1</td>
<td>42.4</td>
<td>42.2</td>
<td>4</td>
<td>1695</td>
<td>0.86</td>
<td>331%</td>
</tr>
<tr>
<td>$\text{Radau5}$</td>
<td>60</td>
<td>70</td>
<td>63</td>
<td>48</td>
<td>57</td>
<td>44</td>
<td>11.3($\times$2)</td>
<td>8782</td>
</tr>
</tbody>
</table>

Table 4: Average Krylov vectors counts and total CPU time. Note: BDF4 and Radau5 list the projections of their first four Newton iterations in columns Proj. 1, Proj. 2, etc.

(b) 2D Advection-Diffusion-Reaction problem with $N = 150^2$

<table>
<thead>
<tr>
<th></th>
<th>Proj. 1</th>
<th>Proj. 2</th>
<th>Proj. 3</th>
<th>Proj. 4</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>Total CPU time</th>
<th>% Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = 0.01$:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{Exp4}$</td>
<td>47.2</td>
<td>42.8</td>
<td>20.1</td>
<td>n/a</td>
<td>3</td>
<td>1101</td>
<td>3.18</td>
<td>100%</td>
</tr>
<tr>
<td>$\text{EpiRK4}$</td>
<td>47.4</td>
<td>40.9</td>
<td>39.7</td>
<td>n/a</td>
<td>3</td>
<td>1280</td>
<td>4.09</td>
<td>129%</td>
</tr>
<tr>
<td>$\text{ERow4}$</td>
<td>45.2</td>
<td>41.0</td>
<td>34.6</td>
<td>n/a</td>
<td>3</td>
<td>1208</td>
<td>3.86</td>
<td>121%</td>
</tr>
<tr>
<td>$\text{BDF4}$</td>
<td>186.0</td>
<td>152.9</td>
<td>121.0</td>
<td>76.6</td>
<td>4.00</td>
<td>3755</td>
<td>33.78</td>
<td>1062%</td>
</tr>
<tr>
<td>$\text{Ros4}$</td>
<td>115.6</td>
<td>113.0</td>
<td>113.4</td>
<td>112.9</td>
<td>4</td>
<td>4549</td>
<td>31.54</td>
<td>992%</td>
</tr>
<tr>
<td>$\text{Radau5}$</td>
<td>147</td>
<td>174</td>
<td>117</td>
<td>138</td>
<td>100</td>
<td>116</td>
<td>88</td>
<td>101</td>
</tr>
</tbody>
</table>

$\text{h = 0.005:}$

<table>
<thead>
<tr>
<th></th>
<th>Proj. 1</th>
<th>Proj. 2</th>
<th>Proj. 3</th>
<th>Proj. 4</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>Total CPU time</th>
<th>% Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Exp4}$</td>
<td>27.3</td>
<td>23.8</td>
<td>12.2</td>
<td>n/a</td>
<td>3</td>
<td>1265</td>
<td>2.71</td>
<td>100.0%</td>
</tr>
<tr>
<td>$\text{EpiRK4}$</td>
<td>27.0</td>
<td>22.2</td>
<td>21.1</td>
<td>n/a</td>
<td>3</td>
<td>1405</td>
<td>3.19</td>
<td>118%</td>
</tr>
<tr>
<td>$\text{ERow4}$</td>
<td>26.3</td>
<td>23.5</td>
<td>19.1</td>
<td>n/a</td>
<td>3</td>
<td>1376</td>
<td>3.08</td>
<td>114%</td>
</tr>
<tr>
<td>$\text{BDF4}$</td>
<td>112.0</td>
<td>93.7</td>
<td>67.1</td>
<td>13.8</td>
<td>3.94</td>
<td>4858</td>
<td>27.88</td>
<td>1029%</td>
</tr>
<tr>
<td>$\text{Ros4}$</td>
<td>54.2</td>
<td>53.1</td>
<td>53.6</td>
<td>53.3</td>
<td>4</td>
<td>4283</td>
<td>15.95</td>
<td>589%</td>
</tr>
<tr>
<td>$\text{Radau5}$</td>
<td>77</td>
<td>91</td>
<td>62</td>
<td>74</td>
<td>52</td>
<td>63</td>
<td>49</td>
<td>59</td>
</tr>
</tbody>
</table>

$\text{h = 0.0025:}$

<table>
<thead>
<tr>
<th></th>
<th>Proj. 1</th>
<th>Proj. 2</th>
<th>Proj. 3</th>
<th>Proj. 4</th>
<th>Projs. per step</th>
<th>Total vectors</th>
<th>Total CPU time</th>
<th>% Exp4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Exp4}$</td>
<td>16.6</td>
<td>14.1</td>
<td>8.1</td>
<td>n/a</td>
<td>3</td>
<td>1552</td>
<td>2.83</td>
<td>100%</td>
</tr>
<tr>
<td>$\text{EpiRK4}$</td>
<td>16.4</td>
<td>12.9</td>
<td>12.0</td>
<td>n/a</td>
<td>3</td>
<td>1652</td>
<td>2.89</td>
<td>102%</td>
</tr>
<tr>
<td>$\text{ERow4}$</td>
<td>16.3</td>
<td>14.1</td>
<td>11.4</td>
<td>n/a</td>
<td>3</td>
<td>1675</td>
<td>3.30</td>
<td>117%</td>
</tr>
<tr>
<td>$\text{BDF4}$</td>
<td>56.9</td>
<td>45.7</td>
<td>25.5</td>
<td>n/a</td>
<td>3.00</td>
<td>4744</td>
<td>15.88</td>
<td>561%</td>
</tr>
<tr>
<td>$\text{Ros4}$</td>
<td>24.4</td>
<td>24.0</td>
<td>24.2</td>
<td>24.1</td>
<td>4</td>
<td>3867</td>
<td>8.33</td>
<td>294%</td>
</tr>
<tr>
<td>$\text{Radau5}$</td>
<td>39</td>
<td>43</td>
<td>31</td>
<td>34</td>
<td>25</td>
<td>27</td>
<td>21</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 4: Average Krylov vectors counts and total CPU time. Note: BDF4 and Radau5 list the projections of their first four Newton iterations in columns Proj. 1, Proj. 2, etc.