UNIVERSITY OF CALIFORNIA, MERCED

Design, Implementation and Performance of Exponential Integrators for High Performance Computing Applications

A dissertation submitted in partial satisfaction of the requirements
for the degree of Doctor of Philosophy

in

Applied Mathematics

by

John Loffeld

Committee in Charge:

Professor Mayya Tokman, Chair

Professor Boaz Ilan

Professor Arnold Kim

Professor Juan Meza

2013

The Dissertation of John Loffeld is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Boaz Ilan

Arnold Kim

Juan Meza

Mayya Tokman Chair

University of California, Merced

2013

Dedicated to my parents

in gratitude for their endless love, support, and encouragement.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost my deepest gratitude goes to my advisor and mentor Mayya Tokman for sparking my interest in applied mathematics, for leading me towards and being instrumental in my admittance into graduate school, for her pivotal help in finding a postdoctoral position, and most of all for her long suffering support and mentorship of me throughout the many years as her student. If I have managed to learn anything about mathematics and about research, I owe it to her. Whatever I have failed to learn is in spite of heroic effort on her part. I am forever indebted to her for guiding and developing me towards a career in applied mathematics.

I also thank my committee members Boaz Ilan, Arnold Kim, and Juan Meza for their advice on my thesis work, for reading drafts of my dissertation, and for their teaching. My gratitude also to Michael Sprague for serving on my committee for several years, and for his helpful guidance in my early years as a graduate student.

My sincere appreciation to Carrie King for her inhumanly competent support of all the graduate students and for her wise counseling during difficult times.

Last but in no way least I thank my parents for their tireless and unwavering support of me over the years. Thank you for your endless love and dedication, and for always encouraging me forward in my career.

The work in Chapter 2 was supported in part by the NSF/DOE Partnership in Plasma Science grant #DMS-0317511 and 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 work in Chapter 3 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 and by the National Science Foundation, Computational Mathematics Program, under Grant No. 1115978.

The work in Chapters 4 and 5 were supported by the National Science Foundation, Computational Mathematics Program, under Grant No. 1115978.

The material of Chapter 2 is reprinted with permission as it appears in Procedia Computer Science 1 (2012) 229-237. The co-author listed in this publication directed and supervised research which forms the basis for the dissertation.

The material of Chapter 3 is reprinted with permission as it appears in Journal of Computational and Applied Mathematics 241 (2013) 45-67. The co-author listed in this publication directed and supervised research which forms the basis for the dissertation.

The material of Chapter 4 is reprinted with permission as it appears in SIAM Journal of Scientific Computing 34 (2012) A2650-A2669, co-authored with Mayya Tokman and Paul Tranquilli. Mayya Tokman directed and supervised research which forms the basis for the dissertation.

Curriculum Vitae

John Loffeld

Ph.D. Candidate - Applied Math
Department of Natural Sciences
University of California, Merced
jloffeld@ucmerced.edu

Objective

Obtain a research position in computational science which utilizes my background in applied mathematics, high performance computing and computer science.

> *Research interests:* Scientific Computing, Numerical Analysis, Parallel Algorithms, Mathematical Modeling.

Education

Ph.D., Applied Mathematics, University of California, Merced, (expected September 2013)

> *Dissertation:* Design, Implementation and Performance of Exponential Integrators for High Performance Computing Applications.
> *Advisor:* Mayya Tokman

B.A., Computer Science, University of California, Berkeley, 1997

Professional Skills

Extensive experience with HPC software and methodologies including

> MPI, OpenMP, GPGPU, Sundials (CVODE).

Proficient in various programming languages including

> C, C++, Fortran, MATLAB, Shell Script, Java, C#, Visual Basic.

Experience

Graduate Research Assistant, University of California, Merced, Fall 2006 - present

Numerical Methods and Algorithms:

- Constructed efficient exponential methods for time integration of large stiff systems of ordinary differential equations.
- Designed optimized exponential propagation iterative integrators of multistep and Runge-Kutta type with adaptive Krylov projection algorithms.
- Developed parallelization strategies for adaptive variable time-step exponential integrators.

Software Development:

- Created serial and parallel implementations of state-of-the-art exponential integrators for high performance computing platforms, such as the 100K+ core Intel Sandy Bridge Stampede, and the 62,976 core AMD Opteron Ranger systems at the Texas Advanced Computing Center.
- Developed a suite of exponential and implicit schemes in MATLAB.

Applications:

- Created a test suite of stiff PDE problems for analyzing performance of exponential and implicit methods on serial and parallel platforms.
- Carried out extensive comparative performance analysis of time integrators for stiff systems of ODEs.
- Applied the developed serial and parallel implementations of exponential integrators to large scale problems in fields such as combustion and plasma physics.

Teaching Assistant, Fall 2006 - Summer 2011

- Taught various undergraduate courses including Numerical Analysis, Linear Algebra, Ordinary Differential Equations, Calculus II, Probability and Statistics

Software Engineer at Lawrence Berkeley National Laboratory, 1998 - 2006

- Developed software for a range of projects related to the energy efficiency of buildings, including automation control software for building components, integration of simulation tools, and analysis of optical properties of glazing systems.
- Led development on multiple large code bases written in various programming languages, particularly C++.
- Co-authored two conference papers and a final project report for the DOE.

Publications

J. Loffeld, M. Tokman, *Implementation of a parallel adaptive-Krylov exponential solver for large scale stiff problems*, in preparation.

J. Loffeld, M. Tokman, *Comparative performance of exponential, implicit, and explicit integrators for stiff systems of ODEs*, Journal of Computational and Applied Mathematics,

241, pp. 45-67, 2013.

M. Tokman, J. Loffeld, and P. Tranquilli, *New adaptive exponential propagation iterative methods of Runge-Kutta type (EPIRK)*, SIAM Journal on Scientific Computing, 34(5), pp. A2650-A2669, 2012.

M. Tokman and J. Loffeld, *Efficient design of exponential-Krylov integrators for large scale computing*, Proceedings of the 10th International Conference on Computational Science, Procedia Computer Science, 1(1), pp. 229-237, 2010.

Presentations

J. Loffeld, M. Tokman, *Design and implementation of exponential integrators*, Lawrence Livermore National Laboratory, June 6, 2013.

J. Loffeld, M. Tokman, *Design and implementation of exponential integrators*, Lawrence Berkeley National Laboratory, April 26, 2013.

J. Loffeld, M. Tokman, *Tailoring exponential integrators for computational efficiency*, Innovative Time Integration Workshop, Innsbruck (Austria), May 2012.

J. Loffeld, M. Tokman, P. Tranquilli, *Adaptive-Krylov exponential propagation iterative methods*, International Congress on Industrial and Applied Mathematics, Vancouver (Canada), July 2011.

Awards

UC Merced Applied Mathematics Dissertation Fellowship.

UC Merced Applied Mathematics Summer 2013 Fellowship.

References

Mayya Tokman, Assistant Professor of Applied Mathematics, School of Natural Sciences, University of California, Merced.

Boaz Ilan, Associate Professor of Applied Mathematics, School of Natural Sciences, University of California, Merced.

Arnold D. Kim, Associate Professor of Applied Mathematics, School of Natural Sciences, University of California, Merced.

Juan Meza, Dean of the School of Natural Sciences, Professor of Applied Mathematics, University of California, Merced.

Abstract

Design, Implementation and Performance
of Exponential Integrators for High Performance Computing
Applications

John Loffeld

A dissertation submitted in partial satisfaction of the requirements for the degree of
Doctor of Philosophy in Applied Mathematics

University of California, Merced

2013

Committee Chair: Mayya Tokman

Exponential integrators have received renewed interest in recent years as a means to approximate stiff systems of ODEs, but are not currently widely used in high performance computing. There have been only limited performance studies comparing them to currently used methods, little work investigating how to optimize their design for computational efficiency, and almost no work on implementing and studying their performance on parallel computers. We present here a detailed performance breakdown and comparison of Krylov-based exponential integrators to each other and to Newton-Krylov implicit solvers, the currently most widely used class of methods for large-scale stiff problems. Our results show exponential integrators perform favorably compared to implicit integrators across a number of different problems. We then introduce a new class of exponential integrators called exponential propagation iterative methods of Runge-Kutta type (EPIRK). Based on our performance analysis we consider some strategies for utilizing their structural features to construct schemes with improved computational efficiency and demonstrate their effectiveness with some numerical experiments. We also describe a parallel implementation of a suite of exponential integrators and give some performance results which show encouraging performance of the methods on problems scaled up to thousands of processors when compared to CVODE, a production-grade parallel implementation of a Newton-Krylov implicit integrators popularly used for high performance computing applications today. We conclude with consideration of possible future research directions.

1 Introduction

Systems of differential equations are characterized as being "stiff" when they constrain explicit numerical integrators to small step sizes in order to maintain numerical stability. Currently implicit methods are typically resorted to due to their better stability properties, despite their considerably higher computational cost per time step. The high cost of implicit methods is due to their need to solve a large system of nonlinear equations each step. Typically such nonlinear systems are treated using Newton's method, which must in turn solve a large linear system each iteration. Krylov iterative methods such as GMRES are the currently favored approach for approximating the solution of large linear systems. Their use in the Newton iteration makes Newton-Krylov implicit methods the most commonly used class of time integrators for large-scale stiff problems today.

Exponential integrators have emerged as a potential alternative to Newton-Krylov implicit methods for approximating large stiff problems. Like implicit methods they have good stability properties, but rather than needing to solve large linear systems, these methods require the evaluation of exponential-like $\varphi_k(A)$ functions of large matrices A, where A is typically the Jacobian matrix or a linear operator from the system. When first introduced in the 1960's [12, 55, 37] exponential integrators were used to solve small systems of only a few variables. The Jacobian matrices were either diagonal, or the exponentiation of the matrices was done using direct methods such as Taylor or Padé approximation. As problems increased in size, though, exponentiating the larger matrices using direct methods became cost prohibitive and attention towards exponential integrators waned. However in the 1980's, Krylov iterative techniques were first used to compute exponentials of symmetric matrices using the Lanczos algorithm [49, 54]. The idea was generalized to exponentials of nonsymmetric matrices by Gallopoulos and Saad [20] and Frienser [19], and finally extended to arbitrary functions of a matrix by Van der Vorst [15]. The use of Krylov techniques for the evaluation of the $\varphi_k(A)$ functions made exponential integrators tractable for large systems, rekindling interest in them. Since then a number of new exponential schemes have been derived [19, 5, 20, 27, 13, 33, 35, 30, 66, 53, 32, 72, 56].

Exponential integrators fall into a number of classes. Integrating Factor (IF) methods were first proposed by Lawson [37] and they work by enacting a change of variables transformation of the system using the integrating factor e^{-tA}, where t is the time variable and A can be any matrix. If A is similar enough to the Jacobian of the original system, the stiffness of the resulting transformed system will be reduced to the point that it can be solved by any classic explicit scheme. The integrating factor approach was generalized by Krogstad to incorporate approximations of the nonlinearities of the system in the transformation, giving the wider class of Generalized Integrating Factor (GIF) methods [35]. Exponential Time Differencing (ETD) schemes are a wide class of methods which are derived from the variation-of-constants integral formulation of the problem through an appropriate numerical approximation of the integral term. Some methods from this class are derived from classical time integrators, such as the exponential W-methods which are derived by replacing the linear systems in Rosenbrock-Wanner (ROW) methods with multiplications by

certain $\varphi_1(A)$ terms [27, 51]. Others are derived directly from the integral formula itself, such as the exponential propagation iterative Runge-Kutta (EPIRK) methods [66], exponential Runge-Kutta methods [29], and the exponential Rosenbrock-type (EROW) methods [30, 9, 32]. Hochbruck and Ostermann review the construction, convergence analysis, and implementation of ETD methods in [31]. Lie group exponential methods employ Lie matrix algebra to ensure that the evolution of the numerical solution lies on the same manifold as the exact solution [14, 17, 44, 48] and are useful for problems for which energy must be conserved or some other invariant must be upheld.

Early work in exponential integrators was largely focused on semilinear problems of the form $y' = Ly + N(y)$, where L and N are linear and nonlinear operators respectively with the stiffness of the problem typically stemming from L [37, 20, 5, 13, 35, 33, 29, 53, 72, 56]. A review of the history of methods formulated for semilinear problems can be found in [45]. Recently attention has increased on constructing exponential methods for general nonlinear problems of the form $y' = f(y)$ [19, 27, 66, 30, 32, 9, 51]. Clearly methods formulated for the latter type of problem can be used to treat problems of the former type, but for problems which can be specified in a semilinear manner it's an ongoing question as to which type of integrator is most efficient for which problems (see [56] for some initial results). This thesis will focus on methods developed for general nonlinear systems as those are applicable to the widest range of scientific problems.

Despite the renewed attention in exponential integrators, much work remains if they are to become widely used on large-scale problems. While there has been increased interest in deriving new schemes, there has been little work on their performance analysis and optimization. While some limited performance results show exponential integrators to be promising for stiff problems, until now there have been no comprehensive studies of the performance of different types of exponential integrators in comparison to each other or to implicit methods. Due to the lack of computational cost analysis, the question of how to optimally tailor the structure of an exponential integrator, as well as how to most efficiently implement a method, is not well understood. Furthermore, the lack of performance studies means the case for using exponential integrators over implicit methods has not been established, either in general or for particular types of problems. Finally, integrators targeted at large-scale problems must be implemented and tuned for massively parallel machines, but there has been almost no work on parallelization of exponential methods. To our knowledge there has been only one study of a parallel implementation of exponential integrators [42]. The method was implemented using Leja point approximation, but comparisons with other integrators were not made. In this thesis we attempt to improve the understanding of the design and performance of exponential integrators through thorough benchmarking and performance analysis, and through the construction and testing of new optimized schemes. The testing is done on both single processor and parallel machines using a software package we wrote called EPIC (ExPonential Integrator Collection), designed for easy extensibility to both new schemes and different approximation algorithms for the evaluation of the $\varphi_k(A)$ terms.

The decisions about how a method should be constructed for good performance depend upon the choice of approximation method for the $\varphi_k(A)$ functions. In this thesis we focus on the use of Krylov projection techniques, due to their good efficiency over a wide range of problems, and because Krylov methods are currently better developed than alternatives. Note though that recently there has been work on other techniques such as polynomial

approximation and contour integral methods [10, 52, 59, 73] which may prove more efficient than Krylov approximation for certain types of problems. See [7, 8] for some early performance comparisons. As our software is agnostic to the type of approximation technique for the $\varphi_k(A)$ terms, in the future we will extend it to include some of these other techniques.

We begin looking at the question of how an exponential method's design impacts its performance in Chapter 2, with the motivation of determining how structural aspects can be exploited in the derivation of more efficient integrators. We introduce the basic structure of exponential integrators in general and detail how Krylov approximation can be used to approximate the products of the $\varphi_k(A)$ matrix functions times a vector which constitute the methods. We note that evaluation of these $\varphi_k(A)v$ terms through Krylov approximation is the principal computational cost of Krylov-based exponential methods and that computational efficiency becomes a matter of (i) minimizing the number of Krylov projections which must be performed each time step and (ii) minimizing the number of Krylov iterations which must be performed each projection. We compare the structure of several exponential schemes and discuss how their differences affect the balance of those two costs. Numerical results are given which illustrate the ideas discussed and demonstrate that the higher rate of convergence of $\varphi_k(A)$ functions with high k value give methods composed of them a marked performance advantage over those composed with low k value. We also present a comparison of the exponential methods with the BDF4 Newton-Krylov implicit method on two benchmark problems and find that the performance of the exponential methods compares well with the implicit method.

In Chapter 3 we continue the discussion of the impact of structure on Krylov cost, now with a much more extensive comparison with Krylov-based implicit integrators. We note that the primary difference between exponential integrators and implicit ones is the type of matrix functions they must evaluate, with exponential integrators requiring the evaluation of the aforementioned $\varphi_k(A)$ functions but implicit methods requiring the computation of matrix rational functions. We argue that Krylov approximation of the $\varphi_k(A)$ functions generally gives a better rate of convergence over the rational functions of a matrix, giving exponential methods an inherent performance advantage over implicit schemes. How a method's overall structure affects its efficiency is a complex matter however, and we present a structural comparison of the exponential schemes with several implicit methods of various design, highlighting how the different aspects affect Krylov efficiency. We then conduct a thorough performance comparison of the exponential and implicit methods on six stiff benchmark problems and give a detailed cost breakdown of each method's performance, discussing the cost tradeoffs between the methods. The comparisons and cost breakdown verifies that the use of the $\varphi_k(A)$ functions does provide exponential integrators with a performance advantage, and demonstrates that exponential methods can perform more efficiently than implicit methods on stiff problems. This work is one of the first detailed numerical studies of how exponential methods perform in comparison to standard integrators.

Continuing in the chapter we describe a new class of exponential methods dubbed exponential propagation iterative methods of Runge-Kutta type (EPIRK), first introduced in [67]. The coefficients of the EPIRK class provide greater flexibility in deriving performance optimized schemes than other known classes, allowing the design of higher order methods with lower Krylov cost compared to previous integrators. As an example of such a method, we describe the new fifth order EPIRK5S3 scheme from the class, also introduced

in [67], and examine how its structure is expected to give streamlined performance. Numerical experiments and a cost breakdown in comparison with other exponential methods are presented to illustrate the ideas. To our knowledge the EPIRK5S3 scheme is the first exponential integrator tailored explicitly around the Krylov cost structure of exponential methods in order to produce a more efficient solver.

In Chapter 3 we also use the results from the performance comparisons to highlight the important concept of Krylov adaptivity. The matrix arguments of the $\varphi_k(A)$ functions are typically the Jacobian matrix of the problem scaled by the current time step. The scaling of the matrices has a large effect on the computational cost of the Krylov approximation, in a manner that is nonlinear in the size of the time step. This results in an unfortunate phenomenon where increasing the step size past a certain pivotal value will result in the computational cost of integration to increase rather than decrease as expected. Particularly for large scale problems, modulating the scaling of the matrix becomes crucial for maintaining efficiency of the integrator. To address this problem, in Chapter 4 we describe the derivation and implementation of two new fifth order schemes from the EPIRK class of schemes, EPIRK5P1 and EPIRK5P2. The methods are constructed specifically to work with a modification of the Krylov adaptivity algorithm described in [50]. The adaptivity algorithm exploits the particular algebra of the $\varphi_k(A)$ functions to decompose their Krylov approximation into a number of cheaper approximations such that the computational cost is minimized. We give some numerical results which demonstrate the large improvements in efficiency of the two methods over previous exponential integrators. The two new methods give a strong example of the value of constructing exponential methods tailored to the particulars of the approximation algorithm being used and are another example of the flexibility of the EPIRK class of methods.

Until now there has been almost no investigation of parallelization of exponential integrators. Chapter 5 describes the implementation of an MPI-based suite of parallel exponential solvers. The software is designed to allow easy extensibility to new schemes and alternate techniques for approximating the $\varphi_k(A)$ functions. The solvers accept problems written for the production Newton-Krylov BDF solver suite CVODE, popularly used for scientific computing on supercomputers today. Compatibility with CVODE will allow practitioners to test exponential integrators without altering the implementation of their problems. Using the new software we describe some early performance results for a parallelized version of the Krylov-adaptive EPIRK5P1 method in comparison to CVODE on a number of unpreconditioned benchmark problems of sizes up to a thousand processors. The performance results show the method to perform well in comparison with CVODE on the problems across all problem sizes. The experiments also demonstrate how increasing problem sizes heightens the importance of managing the computational cost through modulation of the scaling of the matrix, and shows the computational advantage of Krylov adaptivity as a means to maintain scalability. While much work remains to be done to bring exponential integrators into common practice on large parallel machines, this work is to our knowledge the first performance study of exponential methods on the supercomputer scale.

Finally in the conclusions chapter we summarize the results of the thesis and consider some possible future directions of research.

2 Efficient design of exponential-Krylov integrators for large scale computing

2.1 Abstract

As a result of recent resurgence of interest in exponential integrators a number of such methods have been introduced in the literature. However, questions of what constitutes an efficient exponential method and how these techniques compare with commonly used schemes remain to be fully investigated. In this paper we consider exponential-Krylov integrators in the context of large scale applications and discuss what design principles need to be considered in construction of an efficient method of this type. Since the Krylov projections constitute the primary computational cost of an exponential integrator we demonstrate how an exponential-Krylov method can be structured to minimize the total number of Krylov projections per time step and the number of Krylov vectors each of the projections requires. We present numerical experiments that validate and illustrate these arguments. In addition, we compare exponential methods with commonly used implicit schemes to demonstrate their competitiveness.

2.2 Introduction

While the first exponential time integrators were introduced back in the 1960's [12, 55, 37] their popularity among numerical analysts and practitioners has been limited. Initially the main reason for such underutilization was the high computational cost of these schemes. Solving systems of ODEs with an exponential method requires evaluation of a product of an exponential or exponential-type functions of a large matrix with a vector. Even for moderately-sized systems this operation becomes prohibitively expensive if standard techniques such as Taylor or Pade approximations are employed [46]. However, a proposal to use a Krylov projection algorithm for this task significantly reduced computational cost. This idea first appeared in a paper by Nauts and Wyatt [49] where they used Krylov projection to compute exponentials of symmetric matrices that represented discrete Hamiltonian operators, and was later used by Park and Light [54] to exponentially propagate the Schrödinger equation. Van der Vorst extended this idea and proposed to apply Krylov projection to approximate general functions of matrices [15]. A resurgence of interest in exponential methods followed these ideas and a number of such methods have been proposed in the last decade [19, 27, 30, 66, 53].

Coupling exponential methods with the Krylov projection algorithm makes these time integrators much more appealing for large scale computing. Still many questions remain to be answered to enable wide application of these methods to scientific problems. In particular, thorough performance comparisons with state-of-art explicit and implicit integrators are needed and it remains to be demonstrated how details of the design of an exponential integrator affect its performance. This paper presents some results pertaining to the former

question and focuses on the latter issue. We consider exponential integrators as methods that can allow for significant computational savings in integrating large stiff systems of ODEs and from that perspective discuss what constitutes an optimal design of an exponential integrator. The paper is organized as follows. Section 2.3 provides an overview of exponential methods for general nonlinear systems of ODEs and outlines the main features that influence performance of an exponential-Krylov integrator. A suite of test problems is presented in Section 2.4 and the ideas of previous sections are illustrated with numerical examples. Finally, conclusions and directions for future study are presented in Section 2.5.

2.3 Structure of exponential integrators

2.3.1 General derivation and important construction considerations for exponential integrators

In order to illustrate what choices have to be made in the design of an exponential integrator we begin by presenting the general derivation of such schemes. Consider the initial value problem for an autonomous nonlinear system of ODEs

$$y' = f(y), \quad y(t_0) = y_0, \tag{2.1}$$

where $y \in \mathbb{R}^N$. There is no loss of generality in considering an autonomous system since a non-autonomous one can always be converted to the autonomous form by adding the equation $t' = 1$. If the first-order Taylor expansion of $f(y)$ around y_0 exists we can re-write Eq. (2.1) as

$$y' = f(y_0) + f'(y_0)(y - y_0) + r(y) \tag{2.2}$$

with the nonlinear remainder of the first-order Taylor expansion denoted as $r(y) = f(y) - f(y_0) - f'(y_0)(y - y_0)$ and the Jacobian matrix $f'(y_0) \in \mathbb{R}^{N\times N}$. Using the integrating factor $e^{-f'(y_0)t}$ we can find the integral form of the solution to this system at time $t_0 + h$ as

$$y(t_0 + h) = y_0 + \frac{e^{f'(y_0)h} - I}{hf'(y_0)} hf(y_0) + \int_{t_0}^{t_0+h} e^{f'(y_0)(t-t_0)} r(y(t))dt. \tag{2.3}$$

After setting $A_0 = f'(y_0)$ and changing the integration variable to $s = (t - t_0)/h$ in Eq. (2.3) we obtain

$$y(t_0 + h) = y_0 + \frac{e^{hA_0} - I}{hA_0} hf(y_0) + \int_0^1 e^{hAs} hr(y(s))ds. \tag{2.4}$$

Equation (2.4) serves as a starting point in derivation of an exponential method. Alternative derivations are also available, particularly when the nonlinearity is decomposed into the linear and nonlinear terms as $f(y) = Ly + N(y)$ (see [45] for a brief history of exponential methods for such semi-linear problems). However for the general nonlinear systems of type (2.1) which are the focus of this paper, equation (2.4) is a convenient starting point for deriving existing exponential methods by interpreting t_0 as the latest time where an approximate solution is available, considering h as an integration step size and approximating the solution $y(t_0 + h)$.

Constructing an exponential integrator using (2.4) requires accomplishing two tasks: (I) developing an approximation to the nonlinear integral $\int_0^1 e^{hAs} hr(y(s))ds$ and (II) building

an algorithm to evaluate products of functions of matrices and vectors arising from the second term of the right-hand-side of (2.4) and possibly from the approximation chosen for the integral in (I). For example, task (I) can be accomplished by approximating the nonlinear integral using the Runge-Kutta approach. With a two-stage Runge-Kutta-type approximation we can construct the two-stage exponential Runge-Kutta schemes [66]:

$$r_1 = y_0 + a_{11}\varphi_1(\gamma_{11}hA)hf(y_0), \tag{2.5}$$

$$y_1 = y_0 + a_{21}\varphi_1(\gamma_{21}hA)hf(y_0) + a_{22}\varphi_2(\gamma_{22}hA)r(r_1), \tag{2.6}$$

where y_1 is an approximation to the solution $y(t_0+h)$, $\varphi_1(z) = \frac{e^z-1}{z}$, and $\varphi_2(z) = \frac{e^z-1-z}{z^2}$. Choosing $a_{11} = a_{21} = \gamma_{21} = \gamma_{22} = 1$, $\gamma_{11} = 1/2$ and $a_{22} = 2/3$ yields the third-order exponential Runge-Kutta method EPIRK3 proposed in [66]. In general, a polynomial approximation to the nonlinear remainder function $r(y)$ in (2.4) will result in an exponential scheme which approximates the solution as a linear combination of the products of type $\varphi_k(\gamma hA)v_k$ with $v \in \mathbb{R}^N$ and functions $\varphi_k(z)$ defined as

$$\varphi_k(z) = \int_0^1 e^{z(1-s)}\frac{s^{k-1}}{(k-1)!}ds, \qquad k = 0, 1, 2, ... \tag{2.7}$$

Obviously either Runge-Kutta or multistep approaches can be used in the derivation as well as any other construct that yields an approximation to the integral in (2.4). Once a certain ansatz for the approximation to the solution as a linear combinations of terms $a_{lk}\varphi_k(\gamma_{lk}hA)v_{lk}$ is assumed, the order conditions for the coefficients a_{lk}, γ_{lk} can be derived and solved to obtain exponential integrators of the desired order.

After constructing an exponential integrator one needs to address task (II) and to choose an algorithm to approximate the products of functions $\varphi_k(\gamma hA)$ and vectors v_{lk}. For small systems a number of techniques such as Taylor or Pade expansions can be used [46]. If the system size N is large, Krylov projection algorithm becomes the method of choice [58]. Thus a product of a function of a matrix $g(A)$ and a vector v is approximated using projection of the matrix and the vector onto the Krylov subspace $K_m(A, v) = span\{v, Av, ..., A^{m-1}v\}$ as follows. The orthonormal basis $\{v_1, v_2, ..., v_m\}$ of $K_m(A, v)$ is constructed using the modified Arnoldi iteration [3, 58] which can be written in matrix form as

$$AV_m = V_m H_m + h_{m+1,m}v_{m+1}e_m^T \tag{2.8}$$

where $e_m = (0, \ldots, 0, 1, 0, \ldots, 0)^T$ is the unit vector with 1 as the mth coordinate, $\{v_1, v_2, ..., v_m, v_{m+1}\}$ is an orthonormal basis of $K_m(A, b)$, $V_m = [v_1\ v_2\ ...\ v_m] \in \mathbb{R}^{N\times m}$, and

$$H_m = V_m^T A V_m \tag{2.9}$$

is an upper Hessenberg matrix calculated as a side product of the iteration. Matrix $P = V_m V_m^T$ is a projector onto $K_m(A, v)$, thus $g(A)v$ is approximated as a projection

$$g(A)b \approx V_m V_m^T g(A) V_m V_m^T b. \tag{2.10}$$

Recalling Eq. (2.9) and observing that $v_1 = v/\|v\|_2$ we make the final approximation

$$g(A)v \approx \|v\|_2 V_m g(H_m)e_1. \tag{2.11}$$

This algorithm can be used to approximate any of the matrix-function vector products $g(A)v$ with $g(z) = \varphi_k(z)$. It is important to note that the Arnoldi iteration is scale invariant, i.e. once H_m and V_m are calculated for a particular matrix A and vector v, in order to calculate corresponding matrices for γA and v we simply need to scale H_m and V_m by the factor γ, the orthonormal basis does not have to be recalculated from scratch. If $\gamma > 1$ additional Krylov vectors might have to be added to achieve the desired accuracy, if $\gamma < 1$ the approximation with m Krylov vectors will be sufficient. The key to efficiency of the Krylov projection algorithm is keeping the size of the Krylov basis m small so that calculating $g(H_m)$ is cheap and can be done using Pade or any other approximation effective for small matrices. The Krylov basis size m is determined during the course of the iteration using appropriate residuals [57, 27]. Note that m will depend on the eigenvalues of the matrix A, the magnitude of the vector v and the type of function $g(z)$. It has been demonstrated analytically for matrices with a specific spectrum [26] and numerically for some matrices [66] that as k is increased the number of Krylov vectors m required to approximate the product $\varphi_k(A)v$ decreases.

It is clear that approximation of the products $\phi_k(h\gamma_{kl}A_0)v_{lk}$ will constitute the main cost of an exponential scheme since the rest of the required operations is limited to several vector additions and scalar-vector multiplications. Considering efficiency of the complete exponential-Krylov integrator from the perspective of tasks (I) and (II) it is clear that the computational cost of applying an exponential scheme to integration of a large system (2.1) depends on two main features of the chosen method: (i) the total number of products $\phi_k(h\gamma_{kl}A_0)v_{lk}$ that have to be computed and (ii) the number of Krylov vectors that each of these products will require to achieve prescribed accuracy. Thus if we want to construct an exponential integrator of a certain order it is prudent to derive a scheme which minimizes both of these parameters, i.e. requires the minimum possible number of Krylov projections and chooses appropriate functions $g(z)$ and small vectors v so that these projections are fast. Below we consider existing exponential integrators from this point of view and demonstrate how design affects their performance.

2.3.2 Comparing designs of exponential integrators

To illustrate how design affects performance of an exponential-Krylov integrator we consider several existing methods proposed for the solution of general large nonlinear systems (2.1). While the conclusions hold for methods of any order we choose to compare exponential integrators of order four. The first method, *Exp4*, has been developed by Hochbruck et al. [27] and is arguably the most widely known exponential integrator:

$$
\begin{aligned}
&k_1 = \varphi_1(\tfrac{1}{3}hA_0)f(y_0), \quad k_2 = \varphi_1(\tfrac{2}{3}hA_0)f(y_0), \quad k_3 = \varphi_1(hA_0)f(y_0),\\
&w_4 = -\tfrac{7}{300}k_1 + \tfrac{97}{150}k_2 - \tfrac{37}{300}k_3, \quad u_4 = y_0 + hw_4, \quad r_4 = f(u_4) - f(y_0) - hA_0w_4,\\
&k_4 = \varphi_1(\tfrac{1}{3}hA_0)r_4, \quad k_5 = \varphi_1(\tfrac{2}{3}hA_0)r_4, \quad k_6 = \varphi_1(hA_0)r_4,\\
&w_7 = \tfrac{59}{300}k_1 - \tfrac{7}{75}k_2 + \tfrac{269}{300}k_3 + \tfrac{2}{3}(k_4 + k_5 + k_6), \quad u_7 = y_0 + hw_7,\\
&r_7 = f(u_7) - f(y_0) - hA_0w_7,\\
&k_7 = \varphi_1(\tfrac{1}{3}hA_0)r_7,\\
&y_1 = y_0 + h(k_3 + k_4 - \tfrac{4}{3}k_5 + k_6 + \tfrac{1}{6}k_7).
\end{aligned}
\tag{2.12}
$$

Note that due to the invariance of the Arnoldi iteration discussed above, only three Krylov projections are needed - one to approximate k_1, k_2 and k_3, another to estimate k_4, k_5, k_6

and the third one to compute k_7. Each of these projections approximates a product of type $\phi_1(\gamma hA)v$. Note that the function $g(z) = \varphi_1(z)$ does not change, however we can expect that the vector v will decrease in magnitude from one Krylov projection to another if vectors u_i are better approximations to the solution as i increases.

The second method is an exponential propagation iterative Runge-Kutta (*EpiRK*) scheme proposed in [66]:

$$\begin{aligned}
u_1 &= y_0 + a_{11} h\varphi_1(\tfrac{1}{3}hA_0)f(y_0), \\
u_2 &= y_0 + a_{21} h\varphi_1(\tfrac{2}{3}hA_0)f(y_0) + a_{22} h\varphi_2(\tfrac{2}{3}hA_0)r(u_1), \\
y_1 &= y_0 + h\varphi_1(hA_0)f(y_0) + b_1 h\varphi_2(hA_0)r(u_1) + b_2 h[6\varphi_3(hA_0) \\
&\quad - \varphi_2(hA_0)](-2r(u_1) + r(u_2)).
\end{aligned} \tag{2.13}$$

Several methods of third- and forth-order have been derived in [66], in particular, a fourth-order scheme *EpiRK4* with $a_{11} = 3/4, a_{21} = 3/4, a_{22} = 0, b_1 = 160/81, b_2 = 64/81$. The *EpiRK* methods are designed so that the following two principles hold. First, the number of Krylov projections required per time step is minimized by reusing the same vector v in the matrix-function-vector product at each new stage u_i, i.e. here only three Krylov projections must be executed with vectors v in the matrix-function-vector products being $f(y_0)$, $r(u_1)$ and $(-2r(u_1) + r(u_2))$. Second, the number of Krylov vectors each of these projections requires is minimized by having higher order $\varphi_k(z)$ functions that have to be approximated with each new Krylov projection, i.e. $\varphi_1(z)$ for the first projection, $\varphi_2(z)$ for second and $\psi_4(z) = 6\varphi_3(z) - \varphi_2(z)$ for the last Krylov projection.

The last scheme considered here is an exponential Rosenbruck-type scheme *ERow4-1* [32]

$$\begin{aligned}
s_1 &= \varphi_0(\tfrac{1}{2}hA_0)y_0 + \tfrac{1}{2}h\varphi_1(\tfrac{1}{2}A_0)g(y_0), \\
s_2 &= \varphi_0(\tfrac{1}{2}hA_0)y_0 + h\varphi_1(hA_0)g_n(s_1), \\
y_1 &= \varphi_0(hA_0)y_0 + h[\varphi_1(hA_0) - 14\varphi_3(hA_0) + 36\varphi_4(hA_0)]g(y_0) \\
&+ h[16\varphi_3(hA_0) - 48\varphi_4(hA_0)]g(s_1) + h[-2\varphi_3(hA_0) \\
&+ 12\varphi_4(hA_0)]g(s_2),
\end{aligned} \tag{2.14}$$

with $g(y) = f(y) - A_0 y$. In this formulation it appears that *ERow4-1* requires four Krylov projections since terms $\varphi_0(\gamma hA_0)y_0$ must be computed in addition to terms with vectors $g(y_0)$, $g(s_1)$ and $g(s_2)$. However, if we re-write this method in terms of $r(y)$ using the relation $r(y) = g(y) + f(y_0) - A_0 y_0$ we obtain a different formulation of the method we call *ERow4-2*:

$$\begin{aligned}
u_1 &= y_0 + \frac{1}{2}h\varphi_1(\frac{1}{2}hA_0)f(y_0), \\
u_2 &= y_0 + h\varphi_1(hA_0)f(y_0) + h\varphi_1(hA_0)r(u_1), \\
y_1 &= y_0 + h\varphi_1(hA_0)f(y_0) + h[16\varphi_3(hA_0) - 48\varphi_4(hA_0)]r(u_1) \\
&\quad + h[-2\varphi_3(hA_0) + 12\varphi_4(hA_0)]r(u_2)
\end{aligned} \tag{2.15}$$

In this form the method is similar to *EpiRK4* and requires only three Krylov projections per time step. Just as *EpiRK4*, *ERow4-2* uses higher order exponential functions, which we expect to result in faster Krylov convergence for subsequent projections.

2.4 Numerical experiments

In this section we demonstrate how the design of the exponential integrators impacts their performance. We compare constant time step implementations of the three exponential integrators in MATLAB. To illustrate competitiveness of these methods compared to commonly used integrators we include the *BDF4* scheme based on the backwards-differentiation formula of order four and the popular stiff integrator *RADAU5* [23]. For fair comparison both of these methods are implemented using the Krylov projection based algorithm GMRES to solve the linear systems within Newton iterations arising due to implicitness [34].

We have studied the performance of the methods using a suite of test problems (Allen-Cahn [33, 35], Burgers, Brusselator [39, 23], Gray-Scott [21], a semilinear parabolic equation [32], and a nonlinear diffusion equation NILIDI [60]), however for the sake of brevity we choose two representative systems to discuss here. The two-dimensional Allen-Cahn equation and the one-dimensional Burgers equation represent the two end points in the spectrum of problems we studied in terms of how quickly the Krylov projection iteration converges, i.e. to achieve prescribed accuracy the number of Krylov vectors needed per projection is on the order of tens for the Allen-Cahn equation while for the Burgers equation given the same tolerance this number is of the order of a hundred. For convenience we call the former problem "Krylov-easy" and the latter "Krylov-difficult". This terminology directly corresponds to a problem being less or more stiff. Note that all the tests were ran with the same prescribed tolerance for the Krylov projection residuals which was set to 10^{-12}, a value that is smaller then the accuracy requirement for the smallest time step size. Surely this means that the accuracy achieved for some of the Krylov iterations is excessive compared to practical tolerances for given step sizes but such an approach ensures consistent comparison across integrators and helps illustrate the general trends in their performance. Below we describe the two test problems and the parameter values used in the calculations.
Example 2.4.1: The two-dimensional Allen-Cahn equation

$$u_t = u - u^3 + \alpha \nabla^2 u, \quad x, y \in [0, 1] \tag{2.16}$$

with $\alpha = 0.1$ is complemented with the initial and Neumann boundary conditions given by $u = 0.4 + 0.1(x + y) + 0.1 \sin(10x) \sin(20y)$. The diffusive term is discretized with standard second-order finite differences and the problem is integrated over the time interval $t \in [0, 0.1]$.
Example 2.4.2: The one-dimensional Burgers equation

$$u_t = -uu_x + \nu u_{xx}, \quad x \in [0, 1] \tag{2.17}$$

with $\nu = 0.03$ and initial and Dirichlet boundary conditions prescribed using $u = (\sin(3\pi x))^3(1 - x)^{3/2}$. The diffusive term was discretized using the second-order centered finite-differences, the uu_x term was approximated as $uu_x \approx (u_{i+1}^2 - u_{i-1}^2)/(4\Delta x)$, $i = 1, ..., N$ and the problem was integrated over the time interval $t \in [0, 1]$.

Table 2.1 demonstrates how the number of Krylov vectors depends on the structure of an exponential integrator and the effect it has on the overall computational efficiency of the method. As anticipated since both *EpiRK4* and *ERow4-2* use the minimum number of three Krylov projections with the higher order $\varphi_k(z)$ functions for each, the number of Krylov vectors required per projection for these methods is smaller compared to *Exp4* and *ERow4-1*. This is reflected by the total CPU time spent by each of the methods to integrate

the equations over the whole time interval (Tables 2.1). The importance of the reduction in Krylov iterations was particularly pronounced for the more demanding Burgers problem. As can be seen from Table 2.1(b) *EpiRK4* and *ERow4-2* required well less than half the CPU time of *Exp4* at coarse step sizes. Even at the finest step sizes, the savings offered by these two methods were still quite evident. The importance of using the higher order $\varphi_k(z)$ functions can be further illustrated by comparing performance of *ERow4-1* and *Exp4*. Despite the fact that *ERow4-1* has to compute one extra Krylov projection compared to other methods, it still manages to significantly outperform *Exp4* at coarse and medium step sizes. Since adding a vector to the Krylov basis requires orthonormalizing the new vector against every previously computed vector in the basis, the computational cost per vector goes up linearly with the basis size. Therefore the total cost of computing the Krylov basis increases quadratically with the basis size. Thus even modest reduction in the total number of the Krylov vectors per projection can result in significant CPU savings for large basis sizes. As can be seen by comparing with *EpiRK4* and *ERow4-2*, the savings are even greater when both the number of projections is reduced and the falloff of the number of Krylov vectors per projection happens more rapidly.

While the analysis above illustrates the effect of Krylov projections on the computational cost, in order to assess the overall efficiency of a method the accuracy of the final approximation to the solution has to be taken into account. Precision diagrams displayed in Figure 2.1 show the relative performance of the integrators in terms of both accuracy and CPU time required. The problems were each run with several levels of resolution to show how the performance of a method scales with problem size. Figure 2.1 leads to the following conclusions about comparative performance of the methods. First, the effect of the Krylov iterations on efficiency becomes apparent particularly when a problem's stiffness is increased and it becomes more "Krylov-difficult", e.g. as the problem size grows for Allen-Cahn equation from $N = 50^2$ to $N = 150^2$ *EpiRK4* and *ERow4-2* become increasingly more efficient compared to other methods particularly for large step sizes h. Similar behavior can be read off the precision diagrams for the Burgers equation. For example, for step size $h = 0.1$ solution approximations for Burgers equation obtained by *Exp4* and *ERow4* have comparable accuracy, but relative CPU time of *ERow4-2* compared to *Exp4* improves from 60% to 40% as the problem size is increased from $N = 500$ to $N = 1500$. Second, we can see that as the stiffness of a problems is increased the bend in the precision curves particularly for large step sizes indicates that the problem becomes more "Krylov-difficult" and the relative computational cost of the methods becomes more pronounced (note the change in scale of separation between the curves). Note that the bend in the curves illustrates the importance of adaptivity in choosing the dimension of a Krylov subspace. The efficiency of the method is optimal if the tolerance for the residual of a Krylov iteration is calculated depending on the time step size. Our results on development of efficient adaptive algorithms are outside the scope of this paper and will be reported elsewhere. Finally, the figures make it apparent that the exponential methods compete very well with the standard implicit integrators. Note that some of the figures do not include *RADAU5*. The reason for that is the poor performance of this method for these values of h and N which puts it way off scale compared to the other schemes, i.e. the performance curve is so far to the right of the graph that we chose not to include it in the figures in order to preserve clarity in terms of relative performance for the rest of the schemes. In addition to overall computational savings that the exponential methods offer compared to *BDF4* and *RADAU5* we can also

observe that the difficulty in Krylov convergence affects the implicit methods more severely compared to the exponential integrators. For example with the Allen-Cahn equation at step step $h = 0.01$, the CPU cost for *ERow4-2* compared to *BDF4* is about 74% for the smallest problem size. This gap increases to 51% for the largest problem size. The effect is similar but more pronounced for the Krylov-difficult Burgers equation. At $h = 0.01$, the CPU time ratio for *ERow4-2* compared to *BDF4* changes from 12% for the smallest problem size to about 5% for the largest problem size.

Table 2.1: Average Krylov vectors counts and total CPU time.

(a) 2D Allen-Cahn with $N = 150^2$

$h = 0.01$	Average number of Krylov vectors				Total # of Krylov	Total
	Projection 1	Proj. 2	Proj. 3	Proj. 4	vectors	CPU time
Exp4	32.0	25.8	26.7	n/a	84.5	2.48
EpiRK4	27.9	17.4	13.4	n/a	58.7	1.96
ERow4-1	28.8	23.7	23.5	17.1	93.1	2.80
ERow4-2	27.5	19.2	13.7	n/a	60.4	1.97
h = 0.005						
Exp4	20.6	16.2	17.1	n/a	53.8	3.65
EpiRK4	17.4	9.5	6.4	n/a	33.2	3.07
ERow4-1	18.5	14.1	14.1	9.7	56.4	4.30
ERow4-2	17.2	10.4	5.8	n/a	33.4	3.06
h = 0.0025						
Exp4	14.1	10.8	11.3	n/a	36.2	6.54
EpiRK4	11.4	4.6	3.3	n/a	19.4	5.49
ERow4-1	12.5	8.7	8.7	5.7	35.4	7.58
ERow4-2	11.4	5.5	3.3	n/a	20.2	5.44

(b) 1D Burgers with $N = 15000$

$h = 0.01$	Average number of Krylov vectors				Total # of Krylov	Total
	Projection 1	Proj. 2	Proj. 3	Proj. 4	vectors	CPU time
Exp4	133.1	113.2	117.4	n/a	363.7	178.70
EpiRK4	116.7	64.2	42.5	n/a	223.4	73.05
ERow4-1	120.7	107.4	107.4	73.3	408.8	143.09
ERow4-2	114.8	73.3	33.4	n/a	221.5	71.78
h = 0.005						
Exp4	86.4	70.3	72.7	n/a	229.3	84.94
EpiRK4	74.0	32.6	19.7	n/a	126.2	34.71
ERow4-1	76.8	64.3	64.3	40.8	246.2	67.96
ERow4-2	72.9	36.7	14.7	n/a	124.2	33.98
h = 0.0025						
Exp4	57.7	44.1	46.2	n/a	147.9	58.12
EpiRK4	47.0	15.5	10.8	n/a	73.3	26.61
ERow4-1	49.8	37.5	37.5	23.7	148.5	47.82
ERow4-2	46.6	17.0	6.9	n/a	70.4	26.11

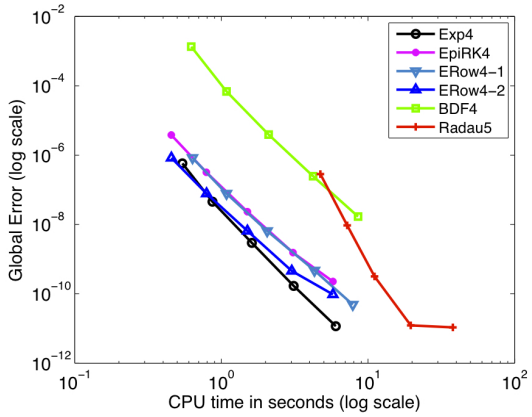
2.5 Conclusions and future work

In this paper we showed how the design of an exponential-Krylov integrator affects its performance. Specifically, we demonstrated that an integrator will be more efficient if it is

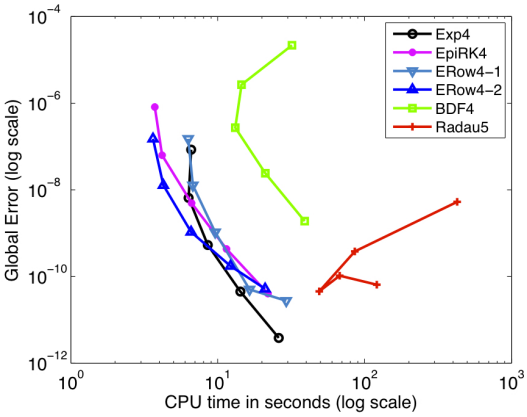
designed to minimize the total number of Krylov projections per time step and the number of Krylov vectors that each of these projections requires. In addition, our studies reveal exponential-Krylov integrators as very competitive alternatives to more commonly used implicit schemes. More detailed studies of the comparative performance of the exponential and implicit schemes both with constant and adaptive time stepping will be presented elsewhere. In addition, we plan to explore the design principles outlined above to construct more optimized exponential-Krylov integrators and study their performance on large-scale scientific applications.

2.6 Acknowledgements

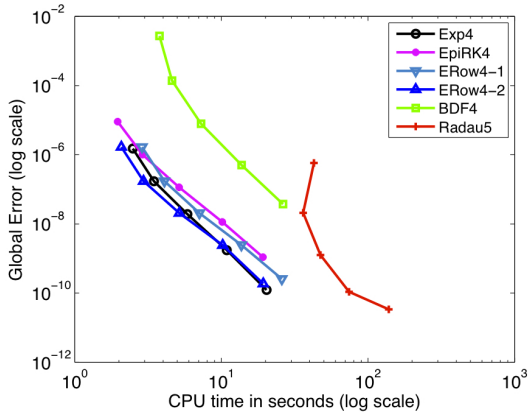
This work was supported in part by the NSF/DOE Partnership in Plasma Science grant #DMS-0317511 and a grant from the U.S. Department of Energy, Office of Science, Offices of Advanced Scientific Computing Research, and Biological & Environmental Research through the U.C. Merced Center for Computational Biology #DE-FG02-04ER25625.



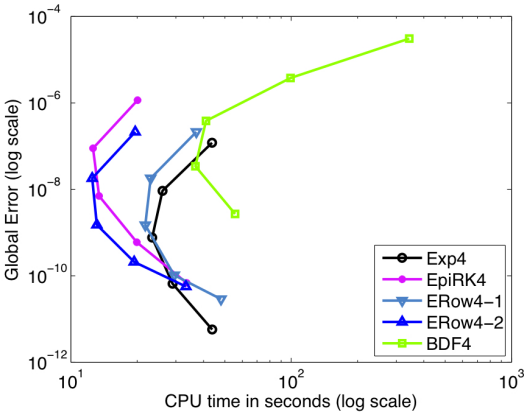
(a) Allen-Cahn with $N = 50^2$



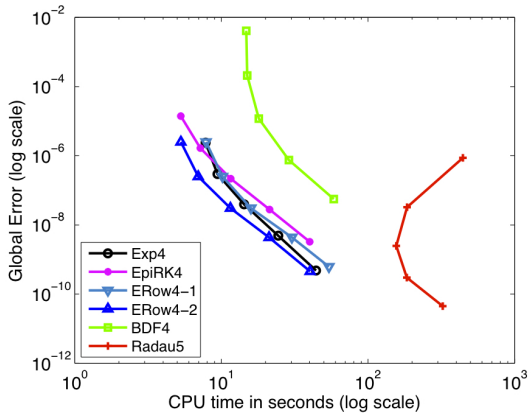
(d) Burgers with $N = 500$



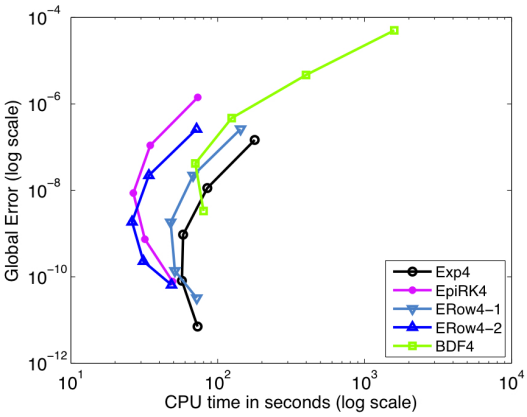
(b) Allen-Cahn with $N = 100^2$



(e) Burgers with $N = 1000$



(c) Allen-Cahn with $N = 150^2$



(f) Burgers with $N = 1500$

Figure 2.1: Precision diagrams for the Allen-Cahn (a-c) and Burgers (d-f) equations for $h = 0.01, 0.005, 0.0025, 0.00125, 0.000625$. Note that the axes scale changes from graph to graph.

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

3.1 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 [68] 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.

3.2 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 for those integrators to be as efficient as possible. We are interested in discovering which types of integrators are most computationally efficient for systems which are very large, stiff, and in general have a nonsymmetric Jacobian.

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 [36, 23]. Instead, stiff problems are usually solved using implicit integrators. Such methods require solutions of an implicit system at 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 [34]. 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 [54, 15]. Since then, a number of exponential integrators for general stiff systems have been proposed [19, 27, 30, 66, 53, 9, 72].

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. While singular performance comparisons of exponential methods can be found in the literature (e.g. [66, 30, 68, 69]), they are rather limited in scope and serve a particular purpose such as, for instance, introducing new integrators and illustrating their properties. Thus these performance results are typically restricted to only a few schemes and/or two or three test problems. Unlike previous publications, from the performance analysis point of view, the present work is more general and focused. Here, comparative performance analysis is presented using a significantly wider range of integrators and test problems. Also, several new higher order methods [68, 69], whose performance has not been studied in detail previously, are considered in both non-adaptive and adaptive form. Additionally, variable time step versions of the state-of-the-art implicit and the best performing exponential methods are also compared. This detailed performance analysis of a spectrum of constant and variable time step, adaptive and nonadaptive, exponential and implicit methods using a range of test problems contributes to the current knowledge of the potential computational advantages offered by exponential integrators. The results presented here also demonstrate what structural features of a method affect performance the most and outline promising research directions that will lead to further optimization and improvement of performance for exponential time integrators. An important objective of this paper is to provide quantitative guidance to a practitioner on the computational savings that can be expected from these schemes.

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 demonstrate how the structure of a method translates to efficiency by presenting the first numerical study of a newly introduced [67, 68] fifth order three-stage exponential propagation iterative (EpiRK) methods designed to optimize performance. These new schemes are studied in both non-adaptive and adaptive form with constant and variable time step selection.

Note that this work addresses integrators for general nonlinear systems of type (3.1). For some problems the nonlinear operator $F(y)$ can be decomposed into a linear stiff part A and a nonlinear nonstiff operator $g(y)$ as $F(y) = Ay + g(y)$. For such systems so-called split exponential methods as well as semi-implicit schemes can be constructed. The main idea behind such techniques is that only the linear part A is exponentiated or solved implicitly. The questions of comparative performance of split vs. nonsplit and split-exponential vs.

semi-implicit schemes are highly dependent on the specifics of a problem and require a separate extensive investigation. While some of the results presented here will be relevant to this topic, it lies outside the scope of this work and will be addressed in detail elsewhere.

The paper is organized as follows. Section 3.3 provides a description of the structure of Krylov-based exponential methods and contrasts them with Newton-Krylov implicit integrators. Section 3.4 describes the methods and problems used for the analysis, and Section 3.5 presents the results of the numerical experiments and explains the performance differences between the methods. To ensure fair comparison, only fourth order exponential integrators are considered in this section. Section 3.6 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 3.7. Finally, Section 3.8 provides conclusions and describes some future research directions.

3.3 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

$$\frac{dy}{dt} = F(y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^N, \tag{3.1}$$

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 (3.1) using a Taylor expansion as

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

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

$$R(y(t+\Delta t)) = F(y(t+\Delta t)) - F(y(t)) - J(y(t))\left[y(t+\Delta t) - y(t)\right]. \tag{3.2}$$

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

$$y(t+\Delta t) = y(t) + (e^{\Delta t J} - I)(J\Delta t)^{-1}\Delta t F(y(t)) + \int_0^1 e^{J\Delta t(1-\theta)} R(y(t+\Delta t\theta))d\theta. \tag{3.3}$$

Equation (3.3) is a starting point for the derivation of most exponential methods. An exponential method is constructed from (3.3) 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

$$\begin{aligned} r_1 &= y_n + h\varphi_1(\tfrac{1}{2}hJ_n)F_n, \\ y_{n+1} &= y_n + h\varphi_1(hJ_n)F_n + \tfrac{2}{3}h\varphi_2(hJ_n)R(r_1), \end{aligned} \tag{3.4}$$

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 φ-functions

$$\varphi_k(z) = \int_0^1 e^{z(1-\theta)} \frac{\theta^{k-1}}{(k-1)!} d\theta, \quad k = 0, 1, 2, \cdots$$

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

$$\varphi_i(hJ) = \sum_{k=0}^{\infty} \frac{(hJ)^k}{(i+k)!}, \tag{3.5}$$

where the zero matrix raised to the zeroth power is considered to be the identity matrix.

Since N is large, computing the products of the φ-functions and vectors (e.g the $\varphi_1(hJ_n)F_n$ term in (3.4)) by algorithms such as Taylor or Padé approximations is computationally prohibitively expensive [46]. Thus we turn to Krylov algorithms which approximate such products by projections onto the Krylov subspace $K_m(hJ, b) = \text{span}\{b, \ldots, (hJ)^{m-1}b\}$.

For general nonsymmetric J, Krylov projection is done using the Arnoldi iteration [57]. 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 $V_m V_m^T$. The upper Hessenberg matrix H_m

$$H_m = V_m^T (hJ) V_m \tag{3.6}$$

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 $V_m V_m^T$ as

$$f(hJ)b \approx V_m V_m^T f(hJ) V_m V_m^T b. \tag{3.7}$$

H_m is then used to evaluate

$$V_m^T f(hJ) V_m \approx f(H_m), \tag{3.8}$$

producing the final approximation

$$f(hJ)b \approx V_m f(H_m) V_m^T b. \tag{3.9}$$

Since the first column vector of V_m is $v_1 = b/||b||_2$, we can use $V_m^T b = ||b||_2 e_1$ to simplify (3.9) as

$$f(hJ)b \approx ||b||_2 V_m f(H_m) e_1. \tag{3.10}$$

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 [46].

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 [26] 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 3.5, 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 [27]:

$$\begin{aligned}
k_1 &= \varphi_1(\tfrac{1}{3}hJ_n)F_n, \quad k_2 = \varphi_1(\tfrac{2}{3}hJ_n)F_n, \quad k_3 = \varphi_1(hJ_n)F_n, \\
w_4 &= -\tfrac{7}{300}k_1 + \tfrac{97}{150}k_2 - \tfrac{37}{300}k_3, \\
u_4 &= y_0 + hw_4, \quad d_4 = F(u_4) - F_n - hJ_0w_4, \\
k_4 &= \varphi_1(\tfrac{1}{3}hJ_n)d_4, \quad k_5 = \varphi_1(\tfrac{2}{3}hJ_n)d_4, \quad k_6 = \varphi_1(hJ_n)d_4, \\
w_7 &= \tfrac{59}{300}k_1 - \tfrac{7}{75}k_2 + \tfrac{269}{300}k_3 + \tfrac{2}{3}(k_4 + k_5 + k_6), \\
u_7 &= y_n + hw_7, \quad d_7 = F(u_7) - F_n - hJ_nw_7, \\
k_7 &= \varphi_1(\tfrac{1}{3}hJ_n)d_7, \\
y_{n+1} &= y_n + h(k_3 + k_4 - \tfrac{4}{3}k_5 + k_6 + \tfrac{1}{6}k_7).
\end{aligned} \tag{3.11}$$

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 (3.9) 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

[66]:

$$
\begin{aligned}
Y_1 &= y_n + a_{11} h\varphi_1(\tfrac{1}{3}hJ_n)F_n, \\
Y_2 &= y_n + a_{21} h\varphi_1(\tfrac{2}{3}hJ_n)F_n + a_{22} h\varphi_2(\tfrac{2}{3}hJ_n)R(Y_1), \\
y_{n+1} &= y_n + h\varphi_1(hJ_n)F_n + b_1 h\varphi_2(hJ_n)R(Y_1) \\
&\quad + b_2 h[6\varphi_3(hJ_n) - \varphi_2(hJ_n)](-2R(Y_1) + R(Y_2)), \\
b_1 &= \frac{96(54 - s^2)(54 - 3s^2 + 2s^3)^2}{729(s^2 + 18)^3}, \\
b_2 &= \frac{64(54 - 3s^2 + 2s^3)^2}{27(s^2 + 18)^3}, \\
a_{11} &= \frac{27(s^2 + 18)}{12(54 - 3s^2 + 2s^3)}, \\
a_{21} &= \frac{3(s^2 + 18)s}{4(54 - 3s^2 + 2s^3)}, \qquad a_{22} = 0.
\end{aligned}
\tag{3.12}
$$

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 [66]. 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 [30]:

$$
\begin{aligned}
Y_1 &= y_n + \tfrac{1}{2} h\varphi_1(\tfrac{1}{2}hJ_n)F_n, \\
Y_2 &= y_n + h\varphi_1(hJ_n)F_n + h\varphi_1(hJ_n)R(Y_1), \\
y_{n+1} &= y_n + h\varphi_1(hJ_n)F_n \\
&\quad + h[16\varphi_3(hJ_n) - 48\varphi_4(hJ_n)]R(Y_1) + h[-2\varphi_3(hJ_n) + 12\varphi_4(hJ_n)]R(Y_2)
\end{aligned}
\tag{3.13}
$$

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 φ-functions are used, though the slower converging $\varphi_1(z)$ function is used in two of the three projections, limiting the benefit of higher-order functions to just the last projection. 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 φ-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 [68].

3.3.1 Comparison with Newton-Krylov implicit integrators

Modern implicit methods for large scale stiff ODE systems employ Krylov projection-based linear solvers [34]. 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 [36, 23]. 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 $f(A)b = e^A b$ is faster than for $f(A)b = (I - A)^{-1}b$ [26]. 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 (3.10) 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 [66] and we will provide further numerical support for this in Section 3.5.

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.

3.4 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 3.5.

3.4.1 Integrators

We compared the exponential integrators Exp4 (3.11), EpiRK4 (3.12), and ERow4 (3.13) 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 [23], 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 [24]. It is possible to use improved algorithms for $\varphi(H_m)$ evaluations from [1, 2]. However, as indicated in section 3.5.2 the total computational cost of $\varphi(H_m)$ evaluations is very small compared to the complexity of the rest of the integrator and will not change the performance data presented here in any significant way. However, for a production code, it is important to note that this portion of the algorithm can be further optimized.

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

$$y_{n+1} = \tfrac{12}{25}hF(t_{n+1}, y_{n+1} + \tfrac{48}{25}y_n - \tfrac{36}{25}y_{n-1} + \tfrac{48}{75}y_{n-2} - \tfrac{3}{25}y_{n-3})$$

is commonly used in modern codes to solve stiff problems [25], 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

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

where

$$\begin{aligned}A &= \begin{pmatrix} \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{pmatrix}, \\ c &= \begin{pmatrix} \frac{4-\sqrt{6}}{10} \\ \frac{4+\sqrt{6}}{10} \\ 1 \end{pmatrix}.\end{aligned}$$

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 [23]. 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 [23]. The general form of a four stage Rosenbrock method is

$$\begin{aligned}(I - h\gamma J)(k_i + \sum_{j=1}^{i-1} \frac{\gamma_{ij}}{\gamma} k_j) &= F(y_{n+1}^{(i)}) + \sum_{j=1}^{i-1} \frac{\gamma_{ij}}{\gamma} k_j, \quad i = 1, .., 4, \\ y_{n+1}^{(i)} &= y_n + h \sum_{j=1}^{i-1} a_{ij} k_j \\ y_{n+1} &= y_n + h \sum_{i=1}^{4} b_i k_i.\end{aligned}$$

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.

Explicit fourth-order Runge-Kutta method RK4: For an explicit method we used the classical

fourth-order Runge-Kutta method given by

$$\begin{aligned} k_1 &= F(t_n, y_n), \\ k_2 &= F(t_n + +\tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1), \\ k_3 &= F(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2), \\ k_4 &= F(t_n + h, y_n + hk_3), \\ y_{n+1} &= y_n + \tfrac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

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.

3.4.2 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. 3.1 and 3.2. 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. 3.5).

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

Allen-Cahn 2D. Two-dimensional Allen-Cahn equation [4]:

$$u_t = \alpha\nabla^2 u + u - u^3,\ x, y \in [-1, 1], t \in [0, 1.0]$$

with $\alpha = 0.1$, using no-flow boundary conditions and initial conditions given by $u = 0.1 + 0.1\cos(2\pi x)\cos(2\pi y)$.

Brusselator 2D. Two-dimensional Brusselator problem [39, 23]:

$$\begin{aligned} u_t &= 1 + uv^2 - 4u + \alpha\nabla^2 u,\ x, y \in [0, 1], t \in [0, 0.1], \\ v_t &= 3u - u^2 v + \alpha\nabla^2 v, \end{aligned}$$

with $\alpha = 0.2$. We used Dirichlet boundary conditions with initial and boundary values given by

$$\begin{aligned} u &= 1 + \sin(2\pi x)\sin(2\pi y), \\ v &= 3. \end{aligned}$$

Burgers. One-dimensional Burgers equation:

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

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

$$uu_x = \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x},\ i = 1, ..., N$$

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

Gray-Scott 2D. Two-dimensional Gray-Scott problem [21]:

$$\begin{aligned} u_t &= d_u \nabla^2 u - uv^2 + a(1-u),\ x,y \in [0,1], t \in [0,0.1], \\ v_t &= d_v \nabla^2 v + uv^2 - (a+b)v, \end{aligned}$$

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

$$\begin{aligned} u &= 1 - e^{-150(x-\frac{1}{2})^2 + (y-\frac{1}{2})^2}, \\ v &= e^{-150(x-\frac{1}{2})^2 + 2(y-\frac{1}{2})^2}. \end{aligned}$$

ADR 2D. Two-dimensional advection-diffusion-reaction equation [9]:

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

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 [61]:

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

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

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

3.5 Numerical experiments: comparisons of fourth order exponential integrators with implicit and explicit schemes.

In this section we describe the results of our comparisons of fourth order 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 integrators, 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.

3.5.1 Quantitative analysis of the integrators performance

It can be seen from the precision diagrams (Fig. 3.1 & 3.2) 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 [58, 33] 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 3.1 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 of 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 φ-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 5.0×10^{-5}.

Table 3.1: Relative CPU time for fixed error tolerances.

(a) Relative performance between Ros4 and Exp4

	Min. difference				Max. difference			
	Error	CPU time			Error	CPU time		
		Exp4	Ros4	% Exp4		Exp4	Ros4	% Exp4
ADR:								
$N = 50^2$	3.9e-06	0.75	1.27	169%	7.6e-02	0.26	0.75	292%
$N = 150^2$	1.5e-06	2.83	7.13	252%	2.5e-03	3.18	15.30	482%
Allen-Cahn:								
$N = 50^2$	5.3e-09	0.57	1.01	178%	3.2e-06	0.21	0.62	298%
$N = 150^2$	9.4e-10	8.79	18.76	213%	3.4e-06	5.89	23.19	394%
Brusselator:								
$N = 50^2$	4.5e-07	0.97	2.32	239%	2.6e-05	0.38	1.38	368%
$N = 150^2$	4.2e-03	10.67	47.20	442%	5.0e-05	8.64	47.22	547%
Burgers:								
$N = 500$	2.0e-11	4.62	7.70	167%	8.4e-08	2.51	7.82	311%
$N = 1500$	9.4e-10	12.89	29.27	227%	6.5e-08	16.82	59.47	353%
Gray-Scott:								
$N = 50^2$	2.0e-07	0.47	0.85	183%	8.2e-06	0.29	0.79	274%
$N = 150^2$	4.5e-08	8.46	18.07	214%	2.7e-05	9.19	25.25	275%
Degenerate Nonlinear Diffusion:								
$N = 500$	2.7e-09	32.15	22.53	70%	1.2e-05	3.86	8.54	221%
$N = 1500$	2.8e-08	88.88	117.39	132%	1.2e-04	22.70	68.00	300%

(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:								
$N = 50^2$	4.1e-10	0.90	1.15	128%	3.5e-07	0.25	1.11	441%
$N = 150^2$	1.3e-10	10.14	111.28	1097%*	1.1e-06	5.78	146.91	2544%*
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%

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 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 φ-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. 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. Additionally, if a problem is stiff enough the performance curves for all methods tend 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 3.5.3. As a result of these two properties of the graphs we can have two "horseshoe"-shaped curves for both the exponential method and BDF4, with the BDF4 curve shifted to the right and up compared to the exponential method graph. If we draw a straight horizontal line representing a fixed level of tolerance, the intersection points with the two graphs will not necessarily correspond to the optimal performance data for either of the methods. For example, if one intersection point is indeed of the "horseshoe"-curve for BDF4 method corresponding to its the optimal performance, since the graph for an exponential method is shifted to the left and down, its intersection point with a straight fixed tolerance line will not be at the optimal tip of the exponential "horseshoe" performance curve. In other words, if tolerance and subsequently the step size are reduced, it is possible to obtain a more accurate solution faster than what the intersection point with a straight line

indicates. 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.

Table 3.2: Relative CPU time for fixed step size h.

(a) Relative performance between BDF4 and Exp4

	Min. difference				Max. difference			
	h	CPU time			h	CPU time		
		Exp4	BDF4	% Exp4		Exp4	BDF4	% Exp4
ADR:								
$N = 50^2$	6.25e-04	0.86	0.93	108%	1.00e-02	0.26	1.08	421%
$N = 150^2$	6.25e-04	4.85	8.27	170%	1.00e-02	3.18	33.78	1063%
Allen-Cahn:								
$N = 50^2$	1.25e-02	0.72	0.61	84%	1.00e-01	0.21	0.29	139%
$N = 150^2$	6.25e-03	10.14	11.33	112%	5.00e-02	5.75	12.52	218%
Brusselator:								
$N = 50^2$	1.25e-03	0.65	0.75	115%	5.00e-03	0.30	0.50	171%
$N = 150^2$	6.25e-04	10.69	16.58	155%	5.00e-03	9.28	20.78	224%
Burgers:								
$N = 500$	6.25e-04	5.41	5.34	99%	1.00e-02	2.51	9.46	377%
$N = 1500$	6.25e-04	12.94	15.13	117%	1.00e-02	19.73	105.84	536%
Gray-Scott:								
$N = 50^2$	1.25e-03	0.60	0.70	116%	5.00e-03	0.28	0.43	154%
$N = 150^2$	1.25e-03	8.82	11.96	136%	1.00e-02	9.19	18.57	202%
Degenerate Nonlinear Diffusion:								
$N = 500$	6.25e-03	34.66	47.06	136%	1.00e-01	3.86	11.48	297%
$N = 1500$	6.25e-03	107.08	165.28	154%	1.00e-01	22.52	100.89	448%

Table 3.2 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 3.2.

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 47 time steps, for which it took 0.036 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 0.036 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 1633 steps, for which it took 26.1 seconds of CPU time. The exponential methods and BDF4 required fewer than 12.5 seconds regardless of how many steps they took. The maximum time required by the exponential methods was 10.1 seconds for Exp4 to compute 160 steps. BDF4 required a maximum of 12.5 seconds for 20 steps (in fact 17 steps since the first three are given as initial conditions), but only required 9.49 seconds for 80 (77) 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 coarsest step size, requiring a maximum of 27.6 seconds for 10 steps, but less than RK4's 26.1 when computing between 20 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 1633 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 [36, 23].

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 φ-functions, Exp4 uses the slower converging φ_1 function for all three of its projections. 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.

3.5.2 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 φ-functions of H_m, and (iv) the right-hand-side function F evaluations. For almost all the computations (i.e. 13 problems $\times$ 5 step sizes $\times$ 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 of the Gray-Scott problem 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 3.3 and 3.4). 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 3.3. 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. 36.6 vectors for Ros4 and 38 for Radau5 compared to just 23.7 for Exp4 for the first projection (first column) at coarse step size. The total number of computed vectors was higher as well, e.g. 1441 for Ros4 and 1755 for Radau5 compared to only 518 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 9.9 for the first, which was only marginally smaller, but 6.4 and 4.1 for the remaining two projections. In the first two Newton iterations, Radau5 computed two projections with 13.5 vectors and two with 9, which were both higher than the corresponding projections of Exp4. The gap in the total number of Krylov vectors was also reduced but remained significant with Ros4 computing 1731 vectors and Radau5 2135 vectors compared to only 813 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 the basis sizes of both projections in each Newton iteration were always larger than that of the corresponding projection in 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 3.1 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 1.54 to 1.63 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 2.78 to 2.88. The inflation in the CPU cost was from 345% of Exp4 to 449%, 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 $2m^2N$ when using modified Gram-Schmidt orthogonalization, and $4m^2N - 4/3m^3$ when using Householder orthogonalization [58]). 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 3.3 we can see the reason for this is that the higher number of vectors taken per projection outweighs the smaller number of projections. For the largest problem size, as the step size decreased to $h = 0.005$ BDF4 required only two Newton iterations. In that case it required 49.7 and 25.3 Krylov vectors for the first and second projections whereas Exp4 required only 25.5 and 18.0 for the first two, but also required 8.4 vectors in a third projection. This balanced out to BDf4 needing 2777 total Krylov vectors compared to 2072 for Exp4, a ratio of 1.34 as many Krylov vectors as Exp4. However, it also needed 1.7 times as much CPU time. The disproportionate increase in CPU time comes from the quadratic scaling of cost with basis size m. The larger basis sizes for BDF4 result in a higher CPU cost per Krylov vector. For most problems BDF4 required only two Newton iterations at all step sizes, yet the larger basis sizes meant that both a higher number of total vectors and greater cost per vector resulted in a bigger overall CPU 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 3.4. 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 3.3 and 3.4 confirm Exp4 takes slightly more Krylov vectors on the second projection compared to EpiRK4, but fewer on the third projection than both EpiRK4 and ERow4, resulting in it having generally the best performance. As an example, for the Allen-Cahn problem with $N = 150^2$ at the coarsest step size, Exp4 used 55.2 vectors for its second projection compared to 50.5 and 54.8 for EpiRK4 and ERow4 respectively. However, it needed only 26.2 vectors for the third projection compared to 41.9 and 41.8 for EpiRK4 and ERow4. In balance, Exp4 computed fewer total Krylov vectors to give about 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%.

3.5.3 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. 3.2). 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 3.3 and 3.4 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 1.5^2 and 2.0^2 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 [27, 50], but so far there is only limited study of how effectively these algorithms find an optimal step size.

3.6 Performance optimization of exponential integrators: efficient fifth order EPIRK methods.

In this section we demonstrate that the performance on exponential integrators can be further improved with careful design of a method. Specifically, we will show that it is possible to construct fifth order EPIRK shemes which have the same per time step computational cost as the fourth order methods described above. While these schemes were originally introduced in [67], their performance was not carefully studied. Here we present a more detailed performance analysis which builds on and further extends the results of the previous sections.

Recently a new class of EpiRK methods have been introduced [67]. The general form of EpiRK schemes is

$$Y_i = y_n + a_{i1}\psi_{i1}(g_{i1}hJ_n)hF_n + \sum_{j=2}^{i-1} a_{ij}\psi_{ij}(g_{ij}hJ_n)h\Delta^{(j-1)}R(y_n), \quad i = 1,..,(s-1)$$

$$y_{n+1} = y_n + b_1\psi_{s1}(g_{s1}hJ_n)hF_n + \sum_{j=2}^{s} b_j\psi_{sj}(g_{ij}hJ_n)h\Delta^{(j-1)}R(y_n), \tag{3.14}$$

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

$$\psi_{ij}(z) = \sum_{k=1}^{s} p_{ijk}\varphi_k(z), \tag{3.15}$$

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,...,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 [67] 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 3.5.

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 (3.14) 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 3.5 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 3.3 and Table 3.6 illustrate how these features of the scheme translate to CPU savings.

Table 3.5: Coefficients of fifth order EpiRK methods.

EpiRK5S3:

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.4165701558065186 & & \\ 0.8624674370127457 & 1.3293114699172297 & \\ 1.0 & 1.1546830340501577 & 0.3093149208665580 \end{bmatrix}$$

$$\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.4165701558065186 & & \\ 0.8624674370127457 & 0.5 & \\ 1.0 & 0.73041615760832766 & 0.32507696706078277 \end{bmatrix}$$

Fig. 3.3 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 3.6 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 $0.33h$ so the small difference is expected. It's worth reiterating that Exp4 uses the φ_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 3.6: Average Krylov vector counts and total CPU time.

(a) 2D Gray-Scott problem with $N = 150^2$

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

3.7 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 affect 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 [75] 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 [74] 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 3.6. This algorithm was implemented using the adaptive Krylov projection algorithm proposed by Niesen and Wright [50]. The detailed description of the adaptive EpiRK methods can be found in [69]. 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 φ-functions-vector products of the form

$$u(t) = \varphi_0(tA)b_0 + \varphi_1(tA)b_1 + \varphi_2(tA)b_2 + ... + \varphi_p(tA)b_p, \quad A \in \mathbb{R}^{N\times N}, \quad b_i \in \mathbb{R}^N, \quad (3.16)$$

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 [63] that $u(t)$ is the solution to the ODE

$$u'(t) = Au(t) + b_1 + tb_2 + ... + \frac{t^{p-1}}{(p-1)!}b_p, \quad u(0) = b_0, \quad (3.17)$$

and if $t_0 = 0 < t_1 < ... < t_k < t_{k+1} < ... < t_{end} = 1$ then values $u(t_i)$ can be computed iteratively using the exact formula

$$u(t_{k+1}) = \varphi_0(\tau_k A)u(t_k) + \sum_{i=1}^{p} \tau_k^i \varphi_i(\tau_k A) \sum_{j=1}^{p-i} \frac{t_k^j}{j!} b_{i+j}, \quad \tau_k = t_{k+1} - t_k. \quad (3.18)$$

The recurrence relation $\varphi_q(A) = \varphi_{q+1}(A)A + \frac{1}{q!}I$ can be employed to express equation (3.18)

in a simplified form

$$u(t_{k+1}) = \tau_k^p \varphi_p(\tau_k A) w_p + \sum_{j=0}^{p-1} \frac{\tau_k^j}{j!} w_j, \tag{3.19}$$

with w_j's computed recursively as

$$w_0 = u(t_k), \quad w_j = A w_{j-1} + \sum_{l=0}^{p-j} \frac{t_k^l}{l!} b_{j+l}, \quad j = 1, ..., p. \tag{3.20}$$

Linear combination (3.16) can then be adaptively computed by stepping equation (3.19) over a set of subintervals $0 = t_0 < t_1 < ... < t_k < t_{k+1} = t_k + \tau_k < ... < t_{end} = 1$ 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 (3.16) with a single large Krylov basis given that the complexity of the Arnoldi iteration scales quadratically with the basis size. The values τ_k in the algorithm are chosen adaptively using error estimates and the cost function, the details of this selection can can be found in [50].

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 3.3, 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 (3.14), this projection minimizing property is equivalent to computing terms associated with coefficients $g_{1j}, g_{2j}, ..., g_{sj}$, with a single Krylov projection, for each j. This feature can be retained in an algorithm with Niesen-Wright adaptivity since the terms $\psi_{1j}(g_{1j}A)b, \psi_{2j}(g_{2j}A)b, ..., \psi_{sj}(g_{sj}A)b$ can be calculated using (3.19) with a single sweep of steps over subinterval $0 = t_0 < t_1 < ... < t_k < t_{k+1} = t_k + \tau_k < ... < t_{end} = 1$ if the functions $\psi_{ij}(z)$ consist of a single $\varphi_k(z)$-function, for some k, and not a linear combination of $\varphi_k(z)$'s. Such approach works as follows. Without loss of generality, let $g_{1j} < g_{2j} < ... < g_{sj}$ and $\psi_{1j}(z) = \psi 2j(z) = .. = \psi sj(z) = \varphi_k(z)$ for some k. Note that in case of a single $\varphi_k(z)$, equation (3.16) reduces to $u(t) = t^k \varphi_k(tA) b_k$. All of the terms $\psi_{1j}(g_{1j}A)b, \psi_{2j}(g_{2j}A)b, ..., \psi_{sj}(g_{sj}A)b$ can then be computed in a single sweep over $0 = t_0 < t_1 < ... < t_k < t_{k+1} = t_k + \tau_k < ... < t_{end} = 1$ by observing that $\varphi_k(g_{ij}A)b$ 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 3.7. 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 3.7: Coefficients of EpiRK5P1.

EpiRK5P1:

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.3512959269505819 & & \\ 0.8440547201165712 & 1.6905891609568963 & \\ 1.0 & 1.27271273173568923 & 2.271459926542262227 \end{bmatrix}$$

$$\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.3512959269505819 & & \\ 0.8440547201165712 & 1.0 & \\ 1.0 & 0.71111095364366870 & 0.62378111953371494 \end{bmatrix}$$

The variable step adaptive EpiRK5P1 and ROWMAP methods were compared on the six problems from section 3.4.2 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 [75], 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. 3.4). 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 $res < 0.1 * h_n * min(Atol, Rtol * ||y_n||)$, where res is the Krylov residual and 0.1 is a safety factor. ROWMAP terminated the Krylov process when $h_n * \sqrt{\frac{1}{N} \sum (\frac{res}{Atol + Rtol * abs(y_n)})^2} < 0.1$.

The results of the comparisons are shown in Fig. 3.4. 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 3.5, 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 [75], while the variable step EpiRK methods with Niesen-Wright adaptivity are a very recent development [69]. The resulting exponen-

tial algorithm can be further optimized and perfected. In particular, the error estimators and the cost functions within the Neisen-Wright adaptivity algorithm can be improved. An illustration of this is the precision diagram for the Burgers equation in Fig. 3.4(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 τ_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. 3.5. 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 τ. 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 approximation 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.

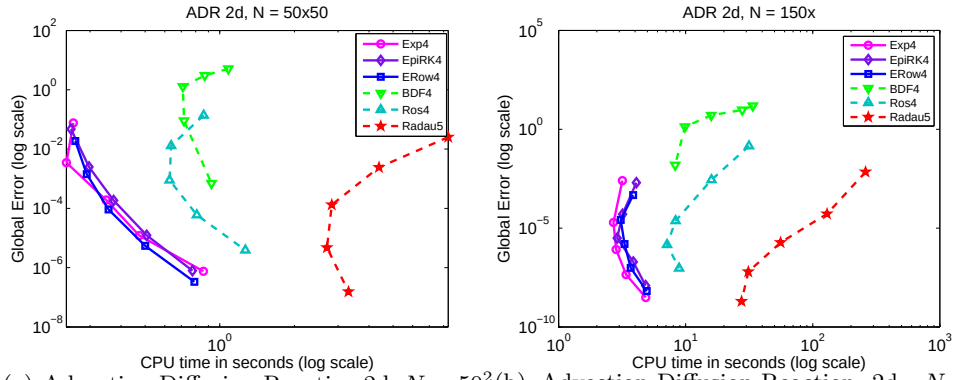
3.8 Conclusions and future work

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 times 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.

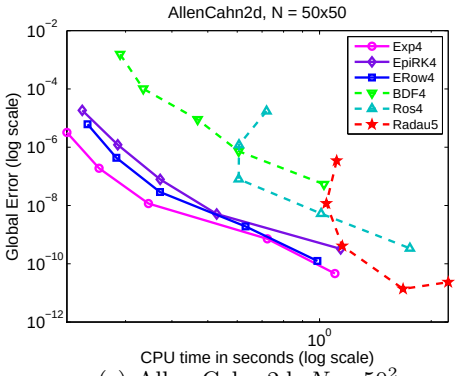
3.9 Acknowledgements

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.

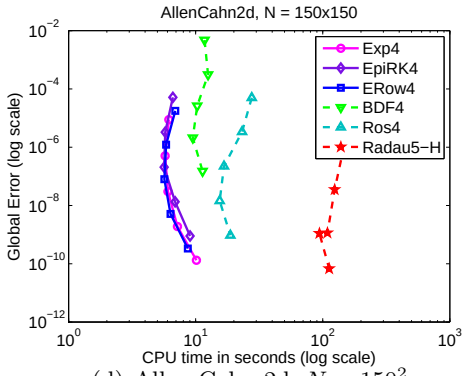


(a) Advection-Diffusion-Reaction 2d, $N = 50^2$

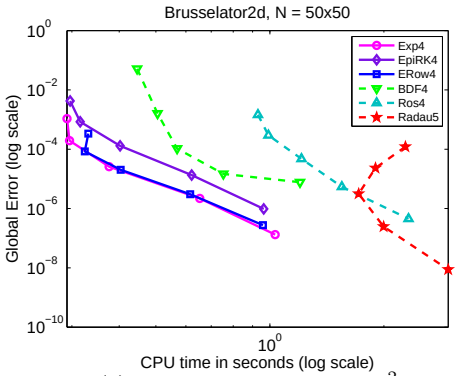
(b) Advection-Diffusion-Reaction 2d, $N = 150^2$



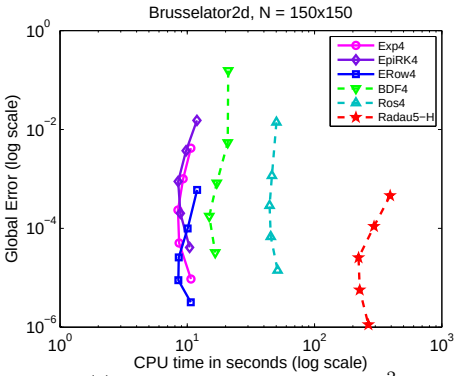
(c) Allen-Cahn 2d, $N = 50^2$



(d) Allen-Cahn 2d, $N = 150^2$

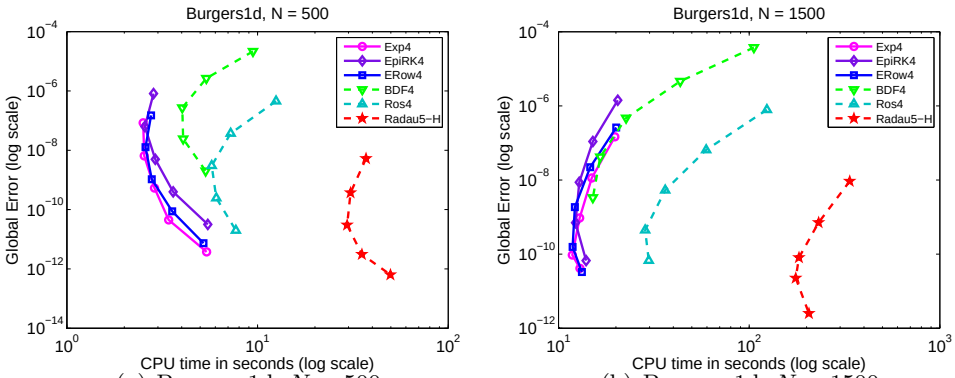


(e) Brusselator 2d, $N = 50^2$



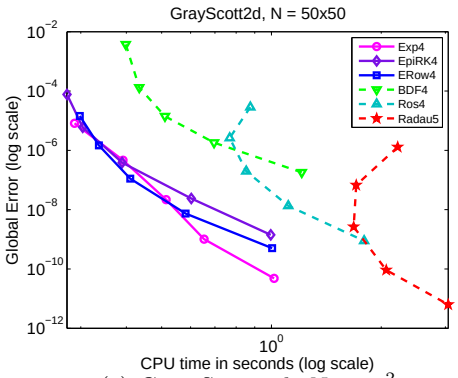
(f) Brusselator 2d, $N = 150^2$

Figure 3.1: Precision diagrams for the Advection-Diffusion-Reaction, Allen-Cahn, and Brusselator problems. Note that the axes scale changes from graph to graph.

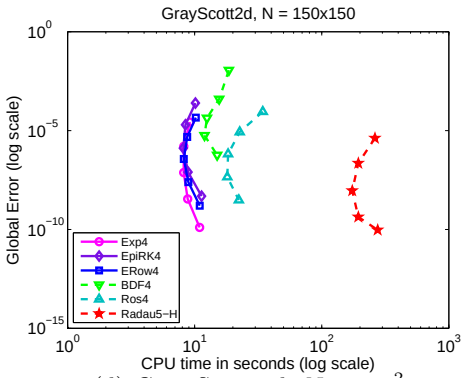


(a) Burgers 1d, $N = 500$

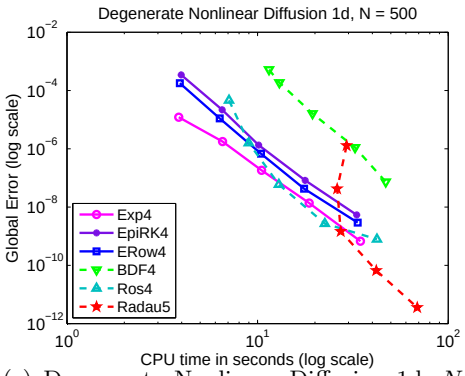
(b) Burgers 1d, $N = 1500$



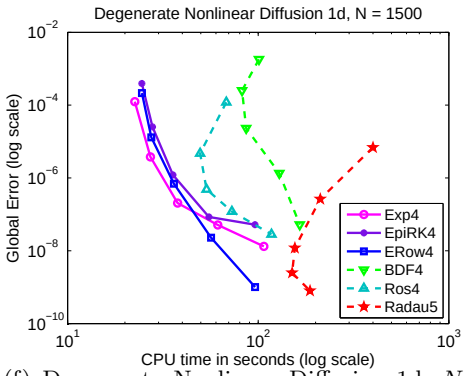
(c) Gray-Scott 2d, $N = 50^2$



(d) Gray-Scott 2d, $N = 150^2$

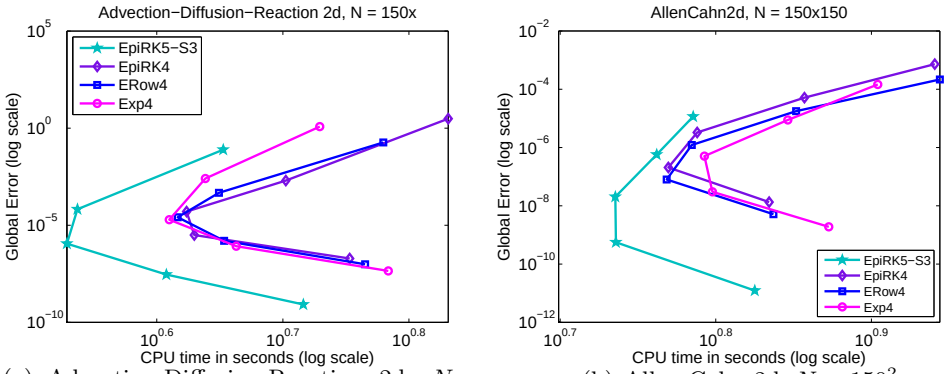


(e) Degenerate Nonlinear Diffusion 1d, $N = 500$



(f) Degenerate Nonlienar Diffusion 1d, $N = 1500$

Figure 3.2: Precision diagrams for the Burgers, Gray-Scott and Degenerate Nonlinear Diffusion problems. Note that the axes scale changes from graph to graph.



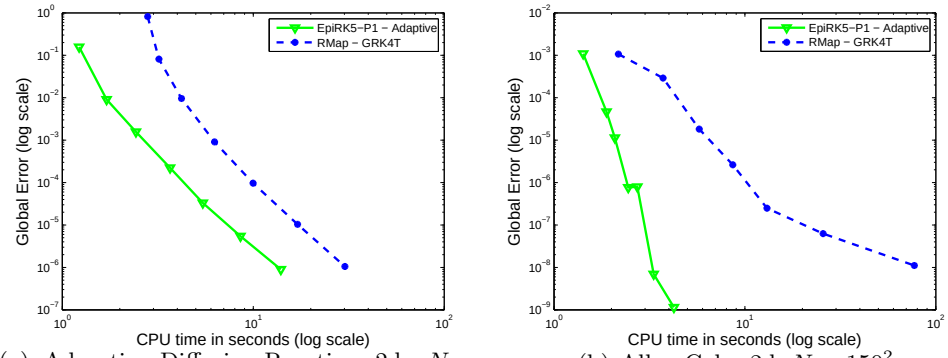
(a) Advection-Diffusion-Reaction 2d, $N = 150^2$

(b) Allen-Cahn 2d, $N = 150^2$

(c) Brusselator 2d, $N = 150^2$

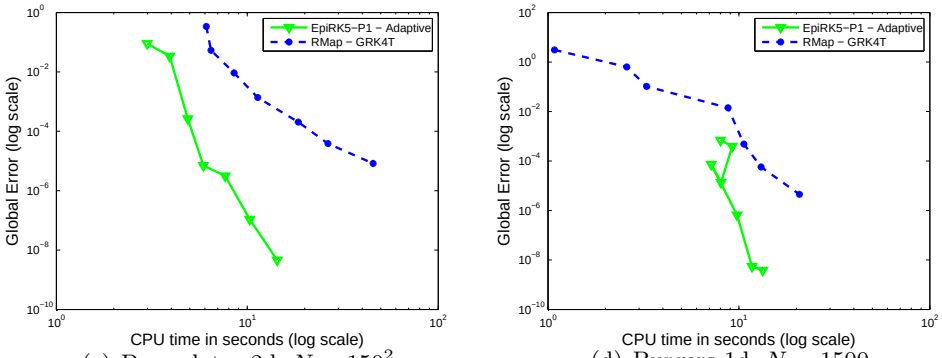
(d) Gray-Scott 2d, $N = 150^2$

Figure 3.3: Precision diagrams comparing the coefficient-optimized EPiRK5S3 method to the other exponential methods.



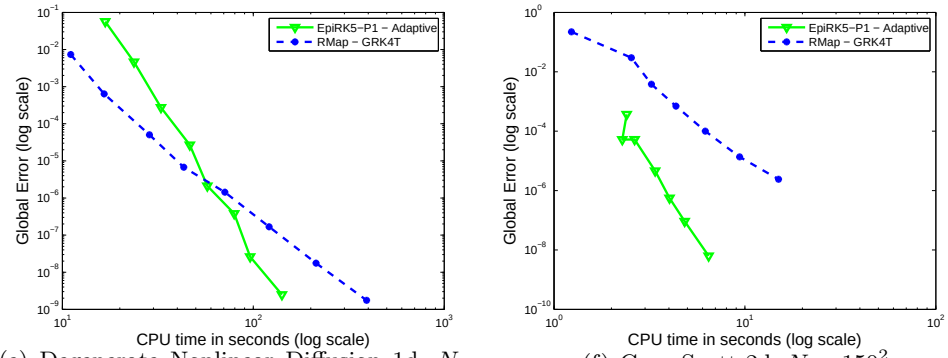
(a) Advection-Diffusion-Reaction 2d, $N = 150^2$

(b) Allen-Cahn 2d, $N = 150^2$



(c) Brusselator 2d, $N = 150^2$

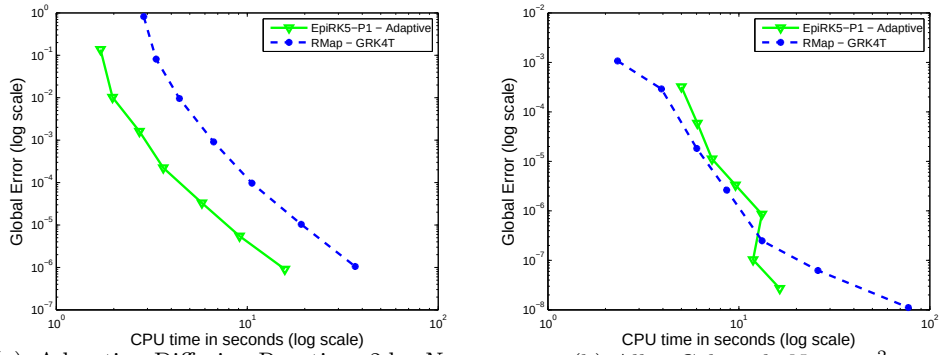
(d) Burgers 1d, $N = 1500$



(e) Degenerate Nonlinear Diffusion 1d, $N = 1500$

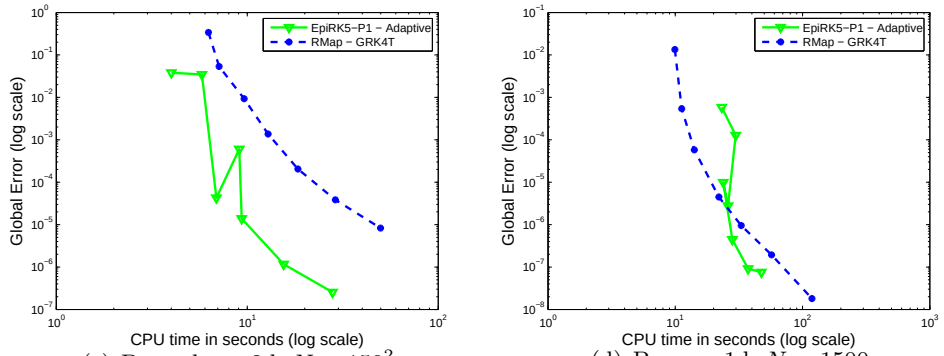
(f) Gray-Scott 2d, $N = 150^2$

Figure 3.4: 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.



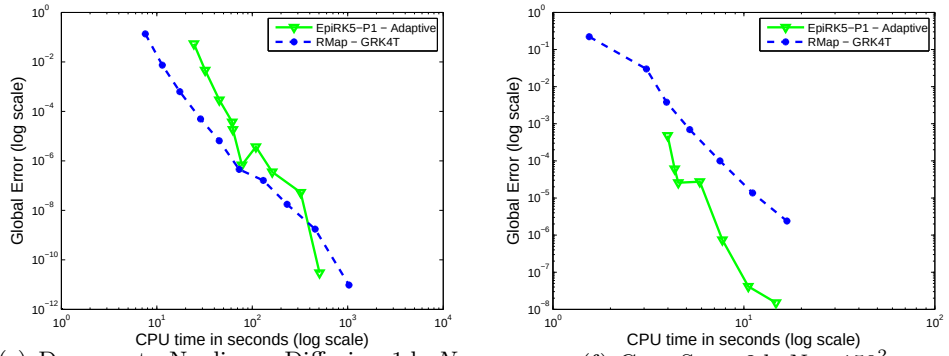
(a) Advection-Diffusion-Reaction 2d, $N = 150^2$

(b) Allen-Cahn 2d, $N = 150^2$



(c) Brusselator 2d, $N = 150^2$

(d) Burgers 1d, $N = 1500$



(e) Degenerate Nonlinear Diffusion 1d, $N = 1500$

(f) Gray-Scott 2d, $N = 150^2$

Figure 3.5: 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 3.3: 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 $N = 50^2$

	Average # of Krylov vectors per step				Projs.	Total	CPU time	
	Proj. 1	Proj. 2	Proj. 3	Proj. 4	per step	vectors	Total	% Exp4
h = 0.02:								
Exp4	23.7	18.6	9.5	n/a	3	518	0.21	100%
EpiRK4	23.1	16.8	14.7	n/a	3	546	0.23	110%
ERow4	22.8	18.6	13.9	n/a	3	553	0.24	114%
BDF4	48.1	32.1	5.1	n/a	3.00	598	0.29	139%
Ros4	36.6	36.2	35.8	35.5	4	1441	0.72	345%
Radau5	38 | 38	26 | 27	16 | 16	8 | 8	4.00(×2)	1755	1.11	533%
h = 0.01:								
Exp4	15.1	10.8	5.6	n/a	3	628	0.25	100%
EpiRK4	15.0	9.7	7.7	n/a	3	646	0.29	112%
ERow4	14.8	10.7	7.0	n/a	3	649	0.28	111%
BDF4	31.2	17.4	2.1	n/a	2.47	843	0.33	132%
Ros4	20.1	19.7	19.7	19.4	4	1576	0.61	238%
Radau5	22 | 22	15 | 15	9 | 8	4 | 4	3.20(×2)	1848	1.04	410%
h = 0.005:								
Exp4	9.9	6.4	4.1	n/a	3	813	0.35	100%
EpiRK4	10.0	5.7	4.4	n/a	3	801	0.37	108%
ERow4	9.9	6.4	4.3	n/a	3	820	0.37	107%
BDF4	17.6	8.9	n/a	n/a	2.00	978	0.47	136%
Ros4	10.9	10.8	10.9	10.7	4	1731	0.61	175%
Radau5	14 | 13	9 | 9	5 | 4	n/a	3.0(×2)	2135	1.15	333%

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

	Average # of Krylov vectors per step				Projs.	Total	CPU time	
	Proj. 1	Proj. 2	Proj. 3	Proj. 4	per step	vectors	Total	% Exp4
h = 0.02:								
Exp4	67.1	55.2	26.2	n/a	3	1485	6.15	100%
EpiRK4	66.2	50.5	41.9	n/a	3	1586	6.62	108%
ERow4	64.5	54.8	41.8	n/a	3	1611	6.92	112%
BDF4	138.0	95.0	18.7	1.0	3.14	1763	11.75	191%
Ros4	109.1	107.4	106.4	104.9	4	4278	27.62	449%
Radau5-H	110 | 111	79 | 80	49 | 49	27 | 27	3.50(×2)	5046	146.91	2388%
h = 0.01:								
Exp4	40.9	31.7	14.6	n/a	3	1744	5.75	100%
EpiRK4	41.0	28.9	20.4	n/a	3	1803	5.76	100%
ERow4	39.9	31.5	21.9	n/a	3	1866	5.86	102%
BDF4	90.3	50.4	4.9	n/a	2.94	2466	12.52	218%
Ros4	64.5	63.1	62.8	62.0	4	5045	23.19	403%
Radau5-H	64 | 64	45 | 45	28 | 28	n/a	3.00(×2)	5492	124.14	2159%
h = 0.005:								
Exp4	25.5	18.0	8.4	n/a	3	2072	6.05	100%
EpiRK4	25.5	16.1	10.9	n/a	3	2100	5.65	93%
ERow4	25.0	17.9	9.4	n/a	3	2091	5.67	94%
BDF4	49.7	25.3	1.0	n/a	2.03	2777	10.28	170%
Ros4	32.8	32.0	32.3	31.8	4	5153	16.70	276%
Radau5-H	39 | 38	27 | 26	15 | 15	n/a	3.00(×2)	6398	108.99	1800%

Table 3.4: 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.

(a) 2D Advection-Diffusion-Reaction problem with $N = 50^2$

	Average # of Krylov vectors per step				Projs.	Total	CPU time	
	Proj. 1	Proj. 2	Proj. 3	Proj. 4	per step	vectors	Total	% Exp4
h = 0.01:								
Exp4	25.9	24.3	13.7	n/a	3	639	0.26	100%
EpiRK4	25.9	22.7	22.2	n/a	3	708	0.25	96%
ERow4	25.2	23.9	20.3	n/a	3	694	0.26	100%
BDF4	73.4	65.9	54.6	31.9	4.00	1580	1.08	415%
Ros4	42.8	42.1	42.4	42.2	4	1695	0.86	331%
Radau5	60 | 70	53 | 63	48 | 57	44 | 53	11.3(×2)	8782	8.34	3208%
h = 0.005:								
Exp4	17.6	16.0	9.6	n/a	3	863	0.24	100%
EpiRK4	17.3	14.9	14.2	n/a	3	926	0.30	125%
ERow4	17.3	16.0	13.6	n/a	3	936	0.29	121%
BDF4	42.5	37.5	26.9	5.8	3.94	1911	0.87	363%
Ros4	23.6	23.5	23.5	23.4	4	1877	0.64	267%
Radau5	35 | 38	30 | 33	27 | 29	23 | 26	7.4(×2)	6957	4.39	1829%
h = 0.0025:								
Exp4	12.5	10.8	7.1	n/a	3	1216	0.35	100%
EpiRK4	12.5	10.0	9.3	n/a	3	1270	0.37	106%
ERow4	12.5	10.8	9.1	n/a	3	1297	0.36	103%
BDF4	23.7	19.8	10.9	1.8	3.11	2022	0.71	203%
Ros4	14.3	14.3	14.4	14.3	4	1834	0.63	180%
Radau5	21 | 21	18 | 17	15 | 14	11 | 11	5.18(×2)	5902	2.83	809%

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

	Average # of Krylov vectors per step				Projs.	Total	CPU time	
	Proj. 1	Proj. 2	Proj. 3	Proj. 4	per step	vectors	Total	% Exp4
h = 0.01:								
Exp4	47.2	42.8	20.1	n/a	3	1101	3.18	100%
EpiRK4	47.4	40.9	39.7	n/a	3	1280	4.09	129%
ERow4	45.2	41.0	34.6	n/a	3	1208	3.86	121%
BDF4	186.0	152.9	121.0	76.6	4.00	3755	33.78	1062%
Ros4	115.6	113.0	113.4	112.9	4	4549	31.54	992%
Radau5	147 | 174	117 | 138	100 | 116	88 | 101	9.3(×2)	8782	260.52	8192%
h = 0.005:								
Exp4	27.3	23.8	12.2	n/a	3	1265	2.71	100.0%
EpiRK4	27.0	22.2	21.1	n/a	3	1405	3.19	118%
ERow4	26.3	23.5	19.1	n/a	3	1376	3.08	114%
BDF4	112.0	93.7	67.1	13.8	3.94	4858	27.88	1029%
Ros4	54.2	53.1	53.6	53.3	4	4283	15.95	589%
Radau5	77 | 91	62 | 74	52 | 63	49 | 59	6.5(×2)	14210	129.60	4782%
h = 0.0025:								
Exp4	16.6	14.1	8.1	n/a	3	1552	2.83	100%
EpiRK4	16.4	12.9	12.0	n/a	3	1652	2.89	102%
ERow4	16.3	14.1	11.4	n/a	3	1675	3.30	117%
BDF4	56.9	45.7	25.5	n/a	3.00	4744	15.88	561%
Ros4	24.4	24.0	24.2	24.1	4	3867	8.33	294%
Radau5	39 | 43	31 | 34	25 | 27	21 | 24	4.7(×2)	10686	55.83	1973%

4 New adaptive exponential propagation iterative methods of Runge-Kutta type (EPIRK)

4.1 Abstract

Exponential integrators have emerged as an efficient alternative to commonly used time-integrators. Recently a new class of exponential propagation iterative methods of Runge-Kutta type (EPIRK) has been introduced [67]. These schemes possess a structure that makes them computationally advantageous compared to other exponential methods. In addition, the general EPIRK formulation offers flexibility that allows derivation of new efficient techniques. In this paper, we use this feature to derive new EPIRK methods which are particularly designed to take advantage of the adaptive Krylov algorithm [50]. The adaptive Krylov method significantly reduces the computational complexity of evaluating products of matrix φ-functions and vectors necessary for implementing an exponential integrator. We present the derivation of the new adaptive EPIRK methods, construct new schemes and illustrate the computational savings they offer using numerical examples.

4.2 Introduction

Recently exponential integrators have emerged as an alternative to standard implicit and explicit techniques for solving large stiff systems of ODEs. While the history of exponential methods dates back to the 1960's [12, 55, 37], construction of efficient exponential schemes for general nonlinear stiff systems is a fairly recent development [37, 20, 13, 5, 33, 35, 19, 27, 66, 30, 53]. It has been demonstrated that exponential schemes can outperform other stiff integrators. In particular, in [41] it was shown that Krylov-exponential propagation iterative methods (EPIRK) can be more efficient than implicit Newton-Krylov schemes. These preliminary results are encouraging, but much research remains to be done to develop efficient exponential algorithms for very large scale problems typically addressed by high-performance computing. Implicit stiff integrators have a long history and many extensions of such schemes have been constructed to overcome practical challenges in solving general and specific large-scale problems. In particular, effective adaptive strategies, which are key to efficiency of a stiff integrator, have been studied in the context of implicit schemes for decades [34]. Practical stiff exponential integrators are at a much earlier stage of development and questions such as adaptivity, error estimators, coupling with spatial discretization and parallelization remain to be fully investigated. In this paper we address the question of adaptivity and propose a new class of adaptive schemes of the exponential propagation iterative Runge-Kutta (EPIRK) type. These techniques possess the computational advantages of the EPIRK methods [67] while employing an adaptive Krylov projection algorithm [50] to further reduce computational cost.

A detailed history and overview of exponential integrators have been presented in previous publications (e.g. [31, 45, 67]). The following brief discussion is intended to provide the

reader with better understanding of how the work presented here fits into the general field of exponential methods. Historically, the first exponential integrators were introduced to solve problems of type $y' = f(y) = Ly + N(y)$, where the stiffness is confined to the linear part Ly of the operator $f(y)$. First examples of such methods date back to 1960's, but research in this field is continuing to this day [37, 18, 13, 5, 33, 28, 53, 43, 16]. Unsplit integrators for a general operator $f(y)$ gained interest in their own right (see [19, 27]. Several formulations to construct general exponential methods of this type were proposed [27, 30, 66] and their performance has been studied and compared to standard explicit and implicit methods [27, 32, 41]; a recent review [31] outlines progress in the field. The main feature of an exponential integrator is that the approximate solution is expressed in terms products of the exponential-like functions of the Jacobian $\varphi_k(A)$ ($A \in \mathbb{R}^{N\times N}$)) and some vectors v ($v \in \mathbb{R}^N$) as explained in the subsequent sections. These evaluations constitute the major computational cost of an exponential method [41]. Thus the efficiency of an exponential method depends on how many of such evaluations are needed and how fast they can be approximated. For general Jacobian matrices, these evaluations are typically done with a Krylov projection algorithm [3]. The class of EPIRK methods was designed to minimize these computational costs allowing derivation of high-order methods with the same complexity as schemes of lower order [66, 68, 67]. However, up to now the proposed EPIRK schemes used a single standard Krylov projection method for each evaluation of a product $\varphi_k(A)v$. Since the computational complexity of a Krylov projection scales quadratically with the number of Krylov vectors it requires, if large Krylov basis size is needed, evaluation of $\varphi_k(A)v$ becomes expensive. Recently an idea of using adaptivity to reduce the cost of these computations was introduced [50]. In this work we utilize the ideas of adaptive Krylov projections and modify the algorithm to combine it with the EPIRK framework. Such an approach allows us to derive new efficient exponential integrators of high-order and better efficiency then previously proposed schemes.

The paper is organized as follows. Section 4.3 provides an introduction to the EPIRK methods and motivates the need for development of adaptive techniques. The main ideas behind the adaptive Krylov algorithm are outlined in section 4.4. The new adaptive EPIRK methods and their derivation are described in section 4.5 along with the ideas underlying their construction. Numerical examples to demonstrate performance of the new schemes are given in section 4.6.

4.3 Background and motivation

Exponential integrators solve general nonlinear stiff systems of ODEs

$$y' = f(y), \quad y(x_0) = y_0, \quad y \in \mathbb{R}^N. \tag{4.1}$$

With the help of an integrating factor $e^{-f'(y_0)x}$ the system (4.1) can be re-written in an integral form

$$y(x_0 + h) = y_0 + h\varphi_1(hA_0)f(y_0) + h\int_0^1 e^{hA(1-\theta)}r(y(x_0 + h\theta))d\theta. \tag{4.2}$$

where $A_0 = f'(y_0) \in \mathbb{R}^{N\times N}$ is the Jacobian matrix, the nonlinear remainder of the first-order Taylor expansion is denoted as $r(y) = f(y) - f(y_0) - f'(y_0)(y - y_0)$ and $\varphi_1(z) =$

$(e^z - 1)/z$ is an analytic function with its matrix-valued form $\varphi_1(hA_0)$ defined via the Taylor series expansion. An exponential integrator is then constructed by choosing an appropriate approximation for the nonlinear integral in (4.2). A polynomial approximation to the nonlinear remainder function $r(y)$ in (4.2) will result in an exponential scheme which computes the solution as a linear combination of the products of type $\varphi_k(\gamma hA)b_k$ with $b_k \in \mathbb{R}^N$ and functions $\varphi_k(z)$ defined as

$$\varphi_k(z) = \int_0^1 e^{z(1-\theta)} \frac{\theta^{k-1}}{(k-1)!} d\theta, \qquad k = 1, 2, ... \tag{4.3}$$

Since approximating terms of type $\varphi_k(\gamma hA)b_k$ is an expensive computation, special care must be taken in developing a quadrature formula for the nonlinear integral in (4.2). EPIRK methods have been introduced to address this issue [66, 67]. These schemes construct a Runge-Kutta type approximation to the nonlinear integral in a way that minimizes both: the number of total required evaluations of $\varphi_k(\gamma hA)b_k$ products and the computational complexity of these evaluations when Krylov projections are used.

The general form of an EPIRK scheme is given as:

$$\begin{aligned} Y_i &= y_0 + a_{i1}\psi_{i1}(g_{i1}hA_0)hf(y_0) + \sum_{j=2}^{i-1} a_{ij}\psi_{ij}(g_{ij}hA_0)h\Delta^{(j-1)}r(y_0), \quad i = 1, .., (s-1) \\ y_1 &= y_0 + b_1\psi_{s1}(g_{s1}hA_0)hf(y_0) + \sum_{j=2}^{s} b_j\psi_{sj}(g_{sj}hA_0)h\Delta^{(j-1)}r(y_0), \end{aligned} \tag{4.4}$$

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

$$\psi_{ij}(z) = \sum_{k=1}^{s} p_{ijk}\varphi_k(z), \tag{4.5}$$

s is the number of stages in a method and the forward differences $\Delta^{(j-1)}r(y_0)$ are computed on the nodes $y_0, Y_1, Y_2, ..., Y_{s-1}$ (recall that for any y, the remainder function can be evaluated as $r(y) = f(y) - f(y_0) - A_0(y - y_0)$ and $r(y_0) = 0$). The coefficients a_{ij}, g_{ij}, b_j and p_{ijk} are chosen based on the order conditions.

The following is one of the structural features of the EPIRK methods that allows one to reduce the computational complexity of an exponential integrator. Some of the most popular and efficient methods to evaluate terms of type $\psi_{ij}(g_{ij}hA_0)v$ (v is a vector) are the Krylov projection-based algorithms [15]. Since the Arnoldi iteration lies at the base of the Krylov projection and it is scale invariant [3], for a fixed j all corresponding terms in (4.4) can be calculated using only one Krylov projection. Thus the total number of Krylov projections required to advance the solution over one time step using (4.4) is equal to the number of stages of the EPIRK method used. We will return to this point in Section 4.5 and use this feature to construct an efficient adaptive exponential method.

An algorithm for solving the order conditions for methods up to order five has been developed and several schemes have been constructed in [67]. It is particularly interesting to note that the EPIRK structure allows derivation of fifth order methods with only three stages. All previously derived exponential integrators with three stages did not exceed order four [27, 30, 32, 31]. The reason such derivation is possible is the flexibility of the order

conditions allowed by the EPIRK formulation. In Section 4.5 we will describe how this property is used to derive adaptivity focused EPIRK methods.

We can illustrate the need for adaptive algorithms by considering precision diagrams for the test problems studied in [41]. For example, consider the two dimensional Allen-Cahn problem [33]:

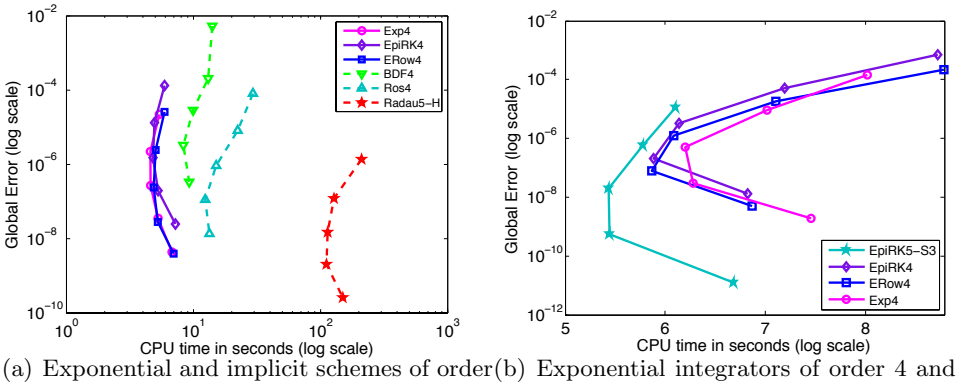
$$u_t \quad = \quad \alpha\nabla^2 u + u - u^3, \; x, y \in [0,1], t \in [0, 0.2] \tag{4.6}$$

with $\alpha = 0.1$ and the Neumann boundary conditions and initial conditions given by $u = 0.4 + 0.1(x + y) + 0.1\sin(10x)\sin(20y)$. Figure 4.1 displays precision diagrams for solving the N-dimensional system of ODEs that results from centered finite-difference discretization of the equation (4.6) on 150 grid points in each spatial dimension (i.e. dimensionality of the system is $N = 150^2$). Figure 4.1(a) displays curves corresponding to solving the system with six methods: three exponential integrators - Exp4 [27], exponential Rosenbrock method ERow4 [32] and EPIRK4 [66] - and three implicit methods - Backward-Differentiation Formula based scheme BDF4, Rosenbrock method Ros4 and Radau5 [23]. Figure 4.1(b) shows only exponential integrators of order four (Exp4, ERow4, EPIRK4) and the fifth-order three stage method EPIRK5-S3. All of the integrators were coupled with the Krylov projection algorithm to approximate terms like $\psi_{ij}(g_{ij}hA_0)v$ for the exponential integrators and terms $(I - \gamma hA_0)^{-1}b_k$ for implicit schemes. All of these methods require three Krylov projections to be executed at each time step. The integrators were compared by picking an initial step size of $h = 0.02$ for all the integrators and successively halving the step size over five sets of computations. 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.

As can be seen from the graphs in Fig. 4.1, all of the precision diagram curves show a bend to the right for large values of h. In other words, it appears that it is actually more computationally efficient to compute with a smaller step size (i.e. h at the start of the curve bends) then with the larger time step. It has been shown in [41] that the cost of the Krylov projection portion of the algorithm is responsible for this fact. The cost of the Krylov projection algorithm is $O(m^2)$, where m is the size of the Krylov subspace, i.e. the number of Krylov vectors computed. If for large time steps the number of Krylov vectors required to achieve a given tolerance grows significantly, the total cost of an integrator will also increase. Thus it is prudent to ask whether it is possible to construct adaptive methods which reduce the Krylov cost and mitigate or eliminate the bend in the precision graphs for large step sizes h. Below we describe how to construct adaptive EPIRK-Krylov methods that improve computational efficiency in this way.

4.4 Adaptive Krylov projection algorithm

As illustrated above, the computational cost of the Krylov algorithm to approximate terms of type $\psi_{ij}(g_{ij}hA_0)v$ depends on the size of the Krylov basis m required to achieve the prescribed accuracy, and scales as $O(m^2)$. Obviously, the size of the Krylov basis m depends on the eigenvalues of A_0, vector v and the values of g_{ij} and h. As h is increased, so is the size of the basis. In fact, for large sizes of the time step h, computing m Krylov vectors might become prohibitively expensive. One strategy to address this problem is to reduce h. However, for a given problem a large h could be perfectly acceptable and, in



(a) Exponential and implicit schemes of order 4.

(b) Exponential integrators of order 4 and 5.

Figure 4.1: Precision diagrams for the 2D Allen-Cahn problem. Note that the axes scale changes from graph to graph.

fact, desirable from the perspective of the accuracy of the overall integrator, especially one which is high order. Thus changing the global h can be an inefficient way to reduce m. An alternative approach would be to develop an algorithm to evaluate $\psi_{ij}(g_{ij}hA_0)v$ separately, outside the global time stepping with h. As part of such a method, h can be scaled within this evaluation, perhaps iteratively, to compute this product with the desired accuracy and efficiency. An example of such an approach would be using the scaling-and-squaring algorithm [46] to compute $\psi_{ij}(g_{ij}hA_0)$. This method, however, is too computationally expensive for large matrices. A more efficient approach has been proposed in [62, 50] and can be summarized as follows.

The goal of the adaptive Krylov algorithm is the evaluation of a linear combination of type

$$\varphi_0(A)b_0 + \varphi_1(A)b_1 + \varphi_2(A)b_2 + ... + \varphi_p(A)b_p, \tag{4.7}$$

where $A \in \mathbb{R}^{N\times N}$ and $b_i \in \mathbb{R}^N$ for $i = 0, ..., p$. Skaflestad and Wright [63] observed that the function

$$u(t) = \varphi_0(tA)b_0 + t\varphi_1(tA)b_1 + t^2\varphi_2(tA)b_2 + ... + t^p\varphi_p(tA)b_p \tag{4.8}$$

is the exact solution of the ODE system

$$u'(t) = Au(t) + b_1 + tb_2 + ... + \frac{t^{p-1}}{(p-1)!}b_p, \quad u(0) = b_0. \tag{4.9}$$

and the expression (4.7) is simply $u(t)$ evaluated at $t = 1$. If the interval $[0, 1]$ is split into subintervals $0 = t_0 < t_1 < ... < t_k < t_{k+1} = t_k + \tau_k < ... < t_K = t_{end} = 1$, solution $u(t_{k+1})$ can be expressed exactly in terms of $u(t_k)$ as

$$u(t_{k+1}) = \varphi_0(\tau_k A)u(t_k) + \sum_{i=1}^{p} \tau_k^i \varphi_i(\tau_k A) \sum_{j=0}^{p-i} \frac{t_k^j}{j!} b_{i+j}, \quad \tau_k = t_{k+1} - t_k. \tag{4.10}$$

Using the recurrence relation $\varphi_q(A) = \varphi_{q+1}(A)A + \frac{1}{q!}$ we can simplify (4.10) to

$$u(t_{k+1}) = \tau_k^p \varphi_p(\tau_k A) w_p + \sum_{j=0}^{p-1} \frac{\tau_k^j}{j!} w_j, \tag{4.11}$$

where w_j's can be computed as

$$w_j = A^j u(t_k) + \sum_{i=1}^{j} A^{j-i} \sum_{l=0}^{j-i} \frac{t_k^l}{l!} b_{i+l}, \quad j = 0, 1, ..., p, \tag{4.12}$$

or recursively via

$$w_0 = u(t_k), \quad w_j = A w_{j-1} + \sum_{l=0}^{p-j} \frac{t_k^l}{l!} b_{j+l}, \quad j = 1, ..., p, \tag{4.13}$$

Clearly only one evaluation of the φ-function product is needed for each step τ_k and this computation involves a matrix scaled by τ_k. Since $0 < \tau_k < 1$, we can expect that evaluating $\varphi_p(\tau_k A)w_p$ requires fewer Krylov vectors than computing $\varphi_p(A)w_p$. Now the challenge is in finding an efficient way to choose τ_k's, $k = 1, ..., K$ so that computing K Krylov subspaces of size m_k is cheaper than calculating one large Krylov subspace for φ-functions evaluated at the unscaled matrix A.

Niesen and Wright [50] developed an algorithm to choose the step sizes τ_k adaptively. They use error estimates for the time-stepping [22] and Krylov projection [57] to construct a cost function $C(\tau_k, m_k)$ (in flops), which helps to determine whether it is more computationally efficient to reduce τ_k or increase the size of the Krylov subspace m_k. The value of τ_k is then chosen so that the error estimation is within the prescribed tolerance and the flops count provided by the cost function is minimized.

To summarize, for a given integration time step h the adaptive Krylov algorithm replaces executing one Krylov projection to approximate the terms of type $\psi_{ij}(g_{ij}hA_0)v$ with several Krylov evaluations of terms $\psi_{ij}(\tau_k g_{ij}hA_0)v$. The terms scaled by τ_k require fewer Krylov basis vectors to achieve prescribed tolerance. The adaptive substepping approach is more efficient if the total computational cost of evaluating the small Krylov subspaces for all K substeps is smaller than computing one large Krylov subspace for the large h. Since the cost of one Krylov projection scales quadratically with the number of Krylov vectors it requires, it is possible that computing a few *small* Krylov bases is computationally cheaper than calculating one *large* Krylov subspace. Numerical examples presented below verify that this property saves computational time compared to non-adaptive algorithms.

The potential computational savings of the adaptive Krylov method can be illustrated using the numerical examples of section 4.6. For each of the problems we extracted a Jacobian matrix J at a particular integration time t and set vector v to be equal to the right-hand-side function of the spatially discretized equation. For a series of step sizes h, we evaluated $\varphi_1(hJ)v$ with the non-adaptive and the Niesen-Wright adaptive [50] implementations of the Krylov algorithm. The 2D problems began with $h = 0.1$ and the 1D Burgers problem with $h = 0.01$. Each successive h was half the size of the previous. Both algorithms were given an accuracy tolerance of 10^{-8}. The residual function used in the Krylov algorithm to estimate the error is somewhat conservative, and in general the non-adaptive implementation produced a result up to an order of magnitude more accurate than the specified

tolerance, whereas the adaptive version produced a result up to two orders of magnitude more accurate than the tolerance. In both cases, the extra accuracy was typically more modest but always better than the given tolerance, with the adaptive algorithm almost always more accurate than the non-adaptive version. Figure 4.2 compares the CPU times of the two Krylov algorithms. It is evident from the graphs that the adaptive algorithm is more efficient. The statistics of the Krylov algorithm presented in table 4.1 makes the advantage of the adaptive Krylov more evident. As we can see, even if the total number of the Krylov vectors computed by adaptive algorithm far exceeds the total number needed by a non-adaptive scheme (e.g. see tbl. 4.1 for Gray-Scott problem with $h = 0.1$), the efficiency of computing smaller Krylov subspaces far outweighs the increase of the total number of vectors computed each integration time step. In the subsequent sections we construct exponential integrators which take advantage of both: the efficient structure of the EPIRK methods and the adaptive Krylov technique.

4.5 New adaptive EPIRK-Krylov methods

Recall that the structure of the EPIRK methods takes advantage of the fact that the Arnoldi iteration is scale invariant. Since each of the stages involves computing terms of type $\psi_p(g_{ip}A_0)b_p$, i.e. where p is fixed and $i = 1, .., s$, the invariance property allows us to approximate all of these terms at the cost of computing only one Krylov basis. To ensure the accuracy of the approximation, we can choose the value of i such that $g_{ip} = \max\limits_{1\le j\le s}\{g_{jp}\}$ and calculate the Krylov basis $S_m = span\{v_1, v_2, ..., v_m\}$ for this term. All of the remaining terms can then be computed by reusing this basis at the expense of calculating for each i the term $\psi_p(g_{ip}H_m)$ for a small matrix H_m obtained as a side product of the Arnoldi iteration. The latter operation can be done via Padé approximation and is a cheap computation compared to the construction of the Krylov basis.

Suppose now we want to evaluate the terms $\psi_p(g_{ip}A_0)b_p$ using an adaptive method outlined in the previous section. In order to preserve the computationally advantageous property that all of these terms for fixed p and $i = 1, ..., s$ are computed with one adaptive Krylov sweep, we will adopt the following strategy. Consider, for example, a general three-stage EPIRK method:

$$\begin{aligned}
Y_1 &= y_0 + a_{11}\psi_{11}(g_{11}hA_0)hf(y_0)\\
Y_2 &= y_0 + a_{21}\psi_{21}(g_{21}hA_0)hf(y_0) + a_{22}\psi_{22}(g_{22}hA_0)hr(Y_1)\\
y_1 &= y_0 + b_1\psi_{31}(g_{31}hA_0)hf(y_0) + b_2\psi_{32}(g_{32}hA_0)hr(Y_1) + b_3\psi_{33}(g_{33}hA_0)h\Delta^2 r(y_0), \quad (4.14)
\end{aligned}$$

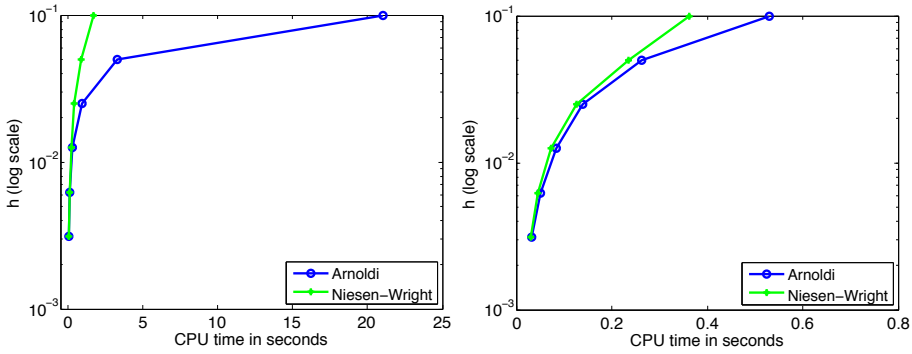
with $\Delta^2 r(y_0) = r(y_0) - 2r(Y_1) + r(Y_2) = -2r(Y_1) + r(Y_2)$ (recall $r(y_0) = 0$) and

$$\psi_{ij}(z) = \sum_{k=1}^{3} p_{ijk}\varphi_k(z). \quad (4.15)$$

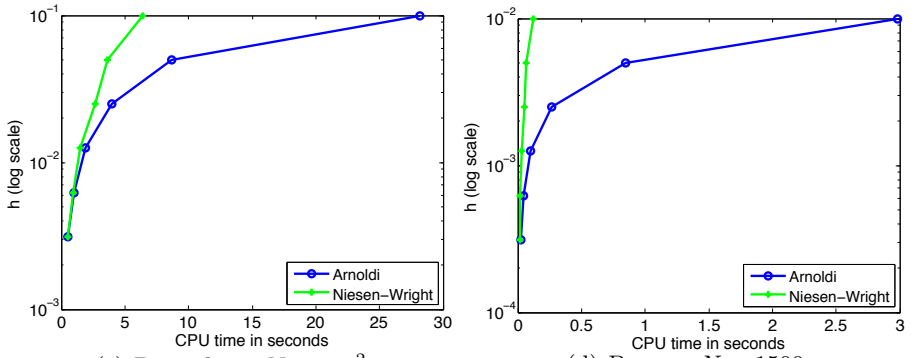
Without loss of generality suppose the coefficients g_{i1} are ordered as $g_{11} \le g_{21} \le g_{31}$. The adaptive Krylov algorithm described above allows computing function

$$u(t) = \varphi_0(tA)b_0 + t\varphi_1(tA)b_1 + t^2\varphi_2(tA)b_2 + ... + t^p\varphi_p(tA)b_p \quad (4.16)$$

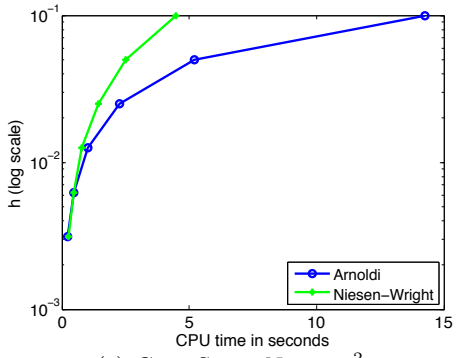
at points $t = t_k$ over some interval $0 = t_0 < t_1 < ... < t_K = t_{end}$ with variable $\tau_k = t_{k+1} - t_k$. If in the EPIRK method we choose $\psi_{11}(z) = \psi_{21}(z) = \psi_{31}(z) = \varphi_p(z)$, set



(a) ADR $N = 150^2$ (b) Allen-Cahn $N = 150^2$



(c) Brusselator $N = 150^2$ (d) Burgers $N = 1500$



(e) Gray-Scott $N = 150^2$

Figure 4.2: Comparison of efficiency of adaptive versus non-adaptive Krylov approximation of $\varphi_1(hJ)v$, where J is a Jacobian from each of the listed problems.

Table 4.1: Performance statistics of non-adaptive and adaptive Krylov algorithms for estimating $\varphi_1(hJ)v$.

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

	Non-adaptive		Adaptive Niesen-Wright			
	# of Krylov vectors	CPU time	# of substeps	# of vectors per substep	Total # of Krylov vectors	CPU time
h = 0.1:	328	21.1	9	55.2	497	1.69
h = 0.05:	195	3.31	5	51.1	255	0.87
h = 0.025:	110	0.92	5	33.8	169	0.43
h = 0.0125:	64	0.30	1	64	64	0.25
h = 0.00625:	39	0.12	1	40	40	0.12
h = 0.003125:	25	0.07	1	26	26	0.06

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

	Non-adaptive		Adaptive Niesen-Wright			
	# of Krylov vectors	CPU time	# of substeps	# of vectors per substep	Total # of Krylov vectors	CPU time
h = 0.1:	90	0.53	5	32.4	162	0.36
h = 0.05:	63	0.26	1	64	64	0.24
h = 0.025:	44	0.14	1	46	46	0.13
h = 0.0125:	32	0.08	1	32	32	0.07
h = 0.00625:	23	0.05	1	23	23	0.04
h = 0.003125:	17	0.03	1	17	17	0.03

(c) 2D Brusselator problem with $N = 150^2$

	Non-adaptive		Adaptive Niesen-Wright			
	# of Krylov vectors	CPU time	# of substeps	# of vectors per substep	Total # of Krylov vectors	CPU time
h = 0.1:	315	28.2	16	50.0	800	6.43
h = 0.05:	226	8.68	9	53.4	481	3.66
h = 0.025:	161	3.95	5	59.8	299	2.64
h = 0.0125:	115	1.91	5	42.8	214	1.46
h = 0.00625:	82	0.99	1	86	86	0.96
h = 0.003125:	59	0.55	1	62	62	0.53

(d) 1D Burgers problem with $N = 1500$

	Non-adaptive		Adaptive Niesen-Wright			
	# of Krylov vectors	CPU time	# of substeps	# of vectors per substep	Total # of Krylov vectors	CPU time
h = 0.01:	235	2.99	14	39.9	559	0.12
h = 0.005:	164	0.84	15	26.9	404	0.07
h = 0.0025:	115	0.27	5	41.4	207	0.05
h = 0.00125:	81	0.10	4	34.0	136	0.03
h = 0.000625:	57	0.04	1	59	59	0.02
h = 0.0003125:	41	0.02	1	42	42	0.02

(e) 2D Gray-Scott problem with $N = 150^2$

	Non-adaptive		Adaptive Niesen-Wright			
	# of Krylov vectors	CPU time	# of substeps	# of vectors per substep	Total # of Krylov vectors	CPU time
h = 0.1:	267	14.2	30	29.1	874	4.47
h = 0.05:	183	5.22	11	37.8	416	2.51
h = 0.025:	124	2.25	12	25.5	306	1.14
h = 0.0125:	83	1.01	5	30.0	150	0.78
h = 0.00625:	55	0.48	1	56	56	0.48
h = 0.003125:	36	0.24	1	36	36	0.27

$A = g_{31}hA_0$ in (4.16), and pick $t_1^* = g_{11}/g_{31}$ and $t_2^* = g_{21}/g_{31}$ where t_1^* and t_2^* are not necessarily equal to any t_k's, then all three terms $\psi_{11}(g_{11}hA_0)hf(y_0)$, $\psi_{21}(g_{21}hA_0)hf(y_0)$ and $\psi_{31}(g_{31}hA_0)hf(y_0)$ can be calculated within one adaptive Krylov sweep of the interval $0 = t_0 < ... < t_{end} = 1$. In this case we are only interested in evaluating a single φ-function $\varphi_p(\tau_k A)b_p$, where $b_p = hf(y_0)$. All other vectors b_i are zero and the formula (4.13) for computing $u(t_{k+1})$ simplifies to

$$w_0 = u(t_k), \quad w_j = Aw_{j-1} + \frac{t_k^{p-j}}{(p-j)!}b_p. \tag{4.17}$$

As we are marching over the interval $0 = t_0 < t_1 < ... < t_n = 1$, once $u(t) = t^p\varphi_p(tA)b_p$ is calculated using formulas (4.11) and (4.17), we can compute $\varphi_p(tA)b_p$ by simply scaling $u(t)$ with t^p. Specifically, to calculate the terms $\psi_{11}(g_{11}hA_0)hf(y_0)$, $\psi_{21}(g_{21}hA_0)hf(y_0)$ and $\psi_{31}(g_{31}hA_0)hf(y_0)$, we need to compute $u(t_1^*)$, $u(t_2^*)$, and $u(1)$, and scale each by t_1^{*p}, t_2^{*p}, and 1 respectively. Ordinarily the adaptivity procedure only computes $u(t)$ at times $t = t_{k+1} = t_k + \tau_k$, where τ_k is chosen adaptively to reduce the computational cost. We can find approximations at times t_1^* and t_2^* by constraining the adaptivity procedure to choose τ_k such that t_1^* and t_2^* are included in the set $\{t_k\}_{k=0}^n$. However, such an algorithm could make us choose τ_k which is not necessarily optimal from the computational complexity point of view. Instead, even if t_1^* and t_2^* are not equal to any t_k chosen by the adaptivity procedure, we can still calculate $u(t_1^*)$ and $u(t_2^*)$ without requiring the computation of any additional Krylov basis beyond those needed for the times t_k. To accomplish this, we use the following approach. Suppose that t_1^* and t_2^* fall between two successive times t_k and t_{k+1} both in the set $\{t_k\}_{k=0}^n$. $u(t_{k+1})$ is computed from $u(t_k)$ by formula (4.11) with $\tau_k = t_{k+1} - t_k$. Note that the same formula can be used to calculate $u(t_i^*)$, $i = 1, 2$, without recomputing the Krylov basis used in computation of $u(t_{k+1})$ if we only replace τ_k with $\tau_i^* = t_i^* - t_k$. Since $t_k < t_i^* < t_{k+1}$ and $\tau_i^* < \tau_k$, the same Krylov basis used for calculation of $u(t_k)$ will yield approximation of equal or better precision to $u(t_i^*)$, $i = 1, 2$.

We can employ a similar procedure to calculate the terms $\psi_{22}(g_{22}hA_0)hr(Y_1)$ and $\psi_{32}(g_{32}hA_0)hr(Y_1)$ using only one adaptive Krylov computation by choosing $\psi_{22}(z) = \psi_{32}(z) = \varphi_p(z)$ and setting $A = \max\{g_{22}, g_{32}\}hA_0$ and ensuring that $u(t)$ is computed at $t = \min\{g_{22}, g_{32}\}/\max\{g_{22}, g_{32}\}$.

In general, an s-stage EPIRK method will require s executions of the adaptive Krylov algorithm. Each adaptive Krylov sweep will calculate all of the terms $\varphi_p(g_{ip}hA_0)b_p$ for fixed p and $i = 1, ..., p$. Note that the term $\psi_{ss}(z)$ is present only in the last stage of an s-stage EPIRK method. Therefore function $\psi_{ss}(z)$ does not have to contain only the term $\varphi_s(z)$ and can be chosen as any linear combination of any $\varphi_j(z)$'s.

Now we will derive the EPIRK methods that allow the adaptive strategy outlined above to work. In [67] we have found the order conditions for general EPIRK methods (4.4) for schemes up to order five and developed an algorithm to systematically solve these order conditions. Table 4.2 lists the order conditions for three-stage EPIRK methods up to order five. To derive the adaptive Krylov-friendly EPIRK methods we prescribe appropriate coefficients p_{ijk} in (4.5) and employ the algorithm of section 4 in [67] to solve the conditions of Table 4.2 using a *Mathematica* script.

First, we constrain the methods (4.4, 4.5) to schemes with $\psi_{ij}(z) = \psi_j(z)$ for any $i = 1, .., s$, i.e. in (4.5) we set $p_{ijk} = p_{jk}$. Then, we define the matrix of coefficients of $\varphi_j(z)$ functions in (4.5) for a three-stage EPIRK method as $P = \{p_{jk}\}_{j,k=1}^s$. We find that if P

is a diagonal matrix the solution to the seventeen order conditions in Table 4.2 does not exist. However, it is possible to solve the order conditions and consequently derive families of EPIRK methods when

$$P = P_1 = \begin{bmatrix} p_{11} & 0 & 0 \\ p_{21} & 0 & 0 \\ 0 & 0 & p_{33} \end{bmatrix} \tag{4.18}$$

and

$$P = P_2 = \begin{bmatrix} p_{11} & 0 & 0 \\ 0 & p_{22} & 0 \\ p_{31} & p_{32} & p_{33} \end{bmatrix}. \tag{4.19}$$

Both of these P matrices lead to the adaptive Krylov-friendly EPIRK schemes since, as we discussed above, the only requirement for a three-stage method is that functions $\psi_{11}(z)$ and $\psi_{22}(z)$ contain only one $\varphi_j(z)$ function.

Solving the order conditions with $P = P_1$ and $P = P_2$ we derive two fifth-order EPIRK methods whose coefficients are listed in Table 4.3. Note that we have obtained families of EPIRK methods rather than just two schemes since the coefficients p_{21}, p_{33} and g_{22} for EPIRK5-P_1 and coefficients p_{22}, g_{22} for EPIRK5-P_2 are arbitrary.

We can verify the order of the methods by applying them to the following simple nonlinear oscillator test problem [6]:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2 \\ -y_1^2 y_2 - y_1 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{4.20}$$

Since the Jacobian matrix is only 2 by 2 in this case, we can use Padé approximation to compute the products of $\psi_j(\gamma A_0)v$. Figure 4.3 shows that the newly constructed methods do, in fact, exhibit the theoretically predicted order. Note that since the new fifth-order integrators have only three stages, their computational complexity is the same as many previously derived fourth-order methods, such as the well-known Exp4 integrator [27].

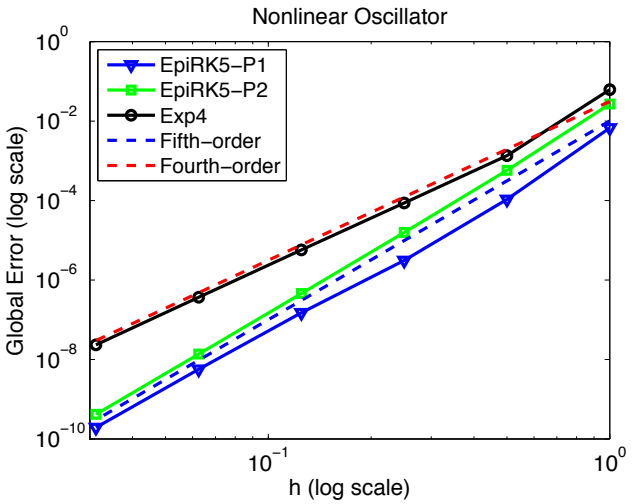


Figure 4.3: Order diagram demonstrating the fifth-order convergence of the EPIRK5-P1 and EPIRK5-P2 methods. The fourth order Exp4 method is included for comparison.

Table 4.2: Order conditions for three-stage EPIRK methods (4.4) up to order five.

Tag Ci	Order conditions for EPIRK methods (4.4) up to order five.
C1	$b_1 p_{11} - 1 = 0$
C2	$b_1 g_{31} p_{11} - 1 = 0$
C3	$6a_{11}^2 b_2 p_{11}^2 p_{21} + 3a_{11}^2 b_2 p_{11}^2 p_{22} - 12a_{11}^2 b_3 p_{11}^2 p_{31} + 6a_{21}^2 b_3 p_{11}^2 p_{31} \ldots$ $- 6a_{11}^2 b_3 p_{11}^2 p_{32} + 3a_{21}^2 b_3 p_{11}^2 p_{32} - 2a_{11}^2 b_3 p_{11}^2 p_{33} + a_{21}^2 b_3 p_{11}^2 p_{33} - 2 = 0$
C4	$b_1 g_{31}^2 p_{11} - 1 = 0$
C5	$12a_{11}^3 b_2 p_{11}^3 p_{21} + 6a_{11}^3 b_2 p_{11}^3 p_{22} - 24a_{11}^3 b_3 p_{11}^3 p_{31} + 12a_{21}^3 b_3 p_{11}^3 p_{31} \ldots$ $- 12a_{11}^3 b_3 p_{11}^3 p_{32} + 6a_{21}^3 b_3 p_{11}^3 p_{32} - 4a_{11}^3 b_3 p_{11}^3 p_{33} + 2a_{21}^3 b_3 p_{11}^3 p_{33} - 3 = 0$
C6	$12a_{11}^2 b_2 g_{11} p_{11}^2 p_{21} + 6a_{11}^2 b_2 g_{11} p_{11}^2 p_{22} - 24a_{11}^2 b_3 g_{11} p_{11}^2 p_{31} + 12a_{21}^2 b_3 g_{21} p_{11}^2 p_{31} \ldots$ $- 12a_{11}^2 b_3 g_{11} p_{11}^2 p_{32} + 6a_{21}^2 b_3 g_{21} p_{11}^2 p_{32} - 4a_{11}^2 b_3 g_{11} p_{11}^2 p_{33} + 2a_{21}^2 b_3 g_{21} p_{11}^2 p_{33} - 3 = 0$
C7	$12a_{11}^2 b_2 g_{32} p_{11}^2 p_{21} + 4a_{11}^2 b_2 g_{32} p_{11}^2 p_{22} - 24a_{11}^2 b_3 g_{33} p_{11}^2 p_{31} + 12a_{21}^2 b_3 g_{33} p_{11}^2 p_{31} \ldots$ $- 8a_{11}^2 b_3 g_{33} p_{11}^2 p_{32} + 4a_{21}^2 b_3 g_{33} p_{11}^2 p_{32} - 2a_{11}^2 b_3 g_{33} p_{11}^2 p_{33} + a_{21}^2 b_3 g_{33} p_{11}^2 p_{33} - 2 = 0$
C8	$b_1 g_{31}^3 p_{11} - 1 = 0$
C9	$30a_{11}^4 b_2 p_{11}^4 p_{21} + 15a_{11}^4 b_2 p_{11}^4 p_{22} - 60a_{11}^4 b_3 p_{11}^4 p_{31} + 30a_{21}^4 b_3 p_{11}^4 p_{31} \ldots$ $- 30a_{11}^4 b_3 p_{11}^4 p_{32} + 15a_{21}^4 b_3 p_{11}^4 p_{32} - 10a_{11}^4 b_3 p_{11}^4 p_{33} + 5a_{21}^4 b_3 p_{11}^4 p_{33} - 6 = 0$
C10	$30a_{11}^3 b_2 g_{11} p_{11}^3 p_{21} + 15a_{11}^3 b_2 g_{11} p_{11}^3 p_{22} - 60a_{11}^3 b_3 g_{11} p_{11}^3 p_{31} + 30a_{21}^3 b_3 g_{21} p_{11}^3 p_{31} \ldots$ $- 30a_{11}^3 b_3 g_{11} p_{11}^3 p_{32} + 15a_{21}^3 b_3 g_{21} p_{11}^3 p_{32} - 10a_{11}^3 b_3 g_{11} p_{11}^3 p_{33} + 5a_{21}^3 b_3 g_{21} p_{11}^3 p_{33} - 6 = 0$
C11	$60a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{21} p_{31} + 30a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{22} p_{31} + 30a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{21} p_{32} \ldots$ $+ 15a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{22} p_{32} + 10a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{21} p_{33} + 5a_{11}^2 a_{21} a_{22} b_3 p_{11}^3 p_{22} p_{33} - 4 = 0$
C12	$30a_{11}^2 b_2 g_{11}^2 p_{11}^2 p_{21} + 15a_{11}^2 b_2 g_{11}^2 p_{11}^2 p_{22} - 60a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{31} + 30a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{31} \ldots$ $- 30a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{32} + 15a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{32} - 10a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{33} + 5a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{33} - 6 = 0$
C13	$30a_{11}^2 b_2 g_{11}^2 p_{11}^2 p_{21} + 15a_{11}^2 b_2 g_{11}^2 p_{11}^2 p_{22} - 60a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{31} + 30a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{31} \ldots$ $- 30a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{32} + 15a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{32} - 10a_{11}^2 b_3 g_{11}^2 p_{11}^2 p_{33} + 5a_{21}^2 b_3 g_{21}^2 p_{11}^2 p_{33} - 6 = 0$
C14	$60a_{11}^3 b_2 g_{32} p_{11}^3 p_{21} + 20a_{11}^3 b_2 g_{32} p_{11}^3 p_{22} - 120a_{11}^3 b_3 g_{33} p_{11}^3 p_{31} + 60a_{21}^3 b_3 g_{33} p_{11}^3 p_{31} \ldots$ $- 40a_{11}^3 b_3 g_{33} p_{11}^3 p_{32} + 20a_{21}^3 b_3 g_{33} p_{11}^3 p_{32} - 10a_{11}^3 b_3 g_{33} p_{11}^3 p_{33} + 5a_{21}^3 b_3 g_{33} p_{11}^3 p_{33} - 6 = 0$
C15	$60a_{11}^2 b_2 g_{11} g_{32} p_{11}^2 p_{21} + 20a_{11}^2 b_2 g_{11} g_{32} p_{11}^2 p_{22} - 120a_{11}^2 b_3 g_{11} g_{33} p_{11}^2 p_{31} \ldots$ $+ 60a_{21}^2 b_3 g_{21} g_{33} p_{11}^2 p_{31} - 40a_{11}^2 b_3 g_{11} g_{33} p_{11}^2 p_{32} + 20a_{21}^2 b_3 g_{21} g_{33} p_{11}^2 p_{32} \ldots$ $- 10a_{11}^2 b_3 g_{11} g_{33} p_{11}^2 p_{33} + 5a_{21}^2 b_3 g_{21} g_{33} p_{11}^2 p_{33} - 6 = 0$
C16	$20a_{11}^2 b_2 g_{32}^2 p_{11}^2 p_{21} + 5a_{11}^2 b_2 g_{32}^2 p_{11}^2 p_{22} - 40a_{11}^2 b_3 g_{33}^2 p_{11}^2 p_{31} + 20a_{21}^2 b_3 g_{33}^2 p_{11}^2 p_{31} \ldots$ $- 10a_{11}^2 b_3 g_{33}^2 p_{11}^2 p_{32} + 5a_{21}^2 b_3 g_{33}^2 p_{11}^2 p_{32} - 2a_{11}^2 b_3 g_{33}^2 p_{11}^2 p_{33} + a_{21}^2 b_3 g_{33}^2 p_{11}^2 p_{33} - 2 = 0$
C17	$b_1 g_{31}^4 p_{11} - 1 = 0$

Table 4.3: Coefficients of fifth-order adaptive Krylov-friendly EPIRK methods.

EPIRK5-P_1

$$p = \begin{bmatrix} 1 & 0 & 0 \\ p_{21} & 0 & 0 \\ 0 & 0 & p_{33} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.35129592695058193092 & & \\ 0.84405472011657126298 & \frac{1.6905891609568963624}{p_{21}} & \\ 1.0 & \frac{1.2727127317356892397}{p_{21}} & \frac{2.2714599265422622275}{p_{33}} \end{bmatrix}$$

$$\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.35129592695058193092 & & \\ 0.84405472011657126298 & g_{22} & \\ 1.0 & 0.71111095364366870359 & 0.62378111953371494809 \end{bmatrix}$$

EPIRK5-P_2

$$p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & p_{22} & 0 \\ -1/3 & -1/3 & 87/10 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.46629408528088195806 & & \\ 0.88217912653363865140 & \frac{2.3790406635847858247}{p_{22}} & \\ 1.0 & \frac{2.1432388712929812169}{p_{22}} & 0.30756483189169759000 \end{bmatrix}$$

$$\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.46629408528088195806 & & \\ 0.88217912653363865140 & g_{22} & \\ 1.0 & 0.92074916488140031449 & 0.79791561832664517267 \end{bmatrix}$$

EPIRK4

$$p = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 0 & -1 & 6 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} \frac{27(s^2+18)}{12(54-3s^2+2s^3)} & & \\ \frac{18s(s^2+18)}{24(54-3s^2+2s^3)} & 0 & \\ 1 & \frac{96(54-s^2)(54-3s^2+2s^3)^2}{729(s^2+18)^3} & \frac{384(54-2s^2+2s^3)^2}{162(s^2+18)^3} \end{bmatrix}, \quad s = \pm\sqrt{30}$$

$$\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 1/3 & & \\ 2/3 & 2/3 & \\ 1 & 1 & 1 \end{bmatrix}$$

4.6 Numerical examples

In this section we demonstrate how adaptivity improves performance of the exponential integrators . We choose several test problems that are routinely used to study performance of stiff integrators (e.g. see [23]). The test problems we selected are:
ADR 2D. Two-dimensional advection-diffusion-reaction equation [9]:

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

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

Allen-Cahn 2D. Two-dimensional Allen-Cahn equation [33]:

$$u_t = \alpha\nabla^2 u + u - u^3,\ x, y \in [-1, 1], t \in [0, 1.0]$$

with $\alpha = 0.1$, using no-flow boundary conditions and initial conditions given by $u = 0.1 + 0.1\cos(2\pi x)\cos(2\pi y)$.

Brusselator 2D. Two-dimensional Brusselator problem [39, 22]:

$$\begin{aligned} u_t &= 1 + uv^2 - 4u + \alpha\nabla^2 u,\ x, y \in [0, 1], t \in [0, 0.1], \\ v_t &= 3u - u^2 v + \alpha\nabla^2 v, \end{aligned}$$

with $\alpha = 0.2$. We used Dirichlet boundary conditions with initial and boundary values given by

$$\begin{aligned} u &= 1 + \sin(2\pi x)\sin(2\pi y), \\ v &= 3. \end{aligned}$$

Burgers. One-dimensional Burgers equation:

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

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 is discretized as

$$uu_x = \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x},\ i = 1, ..., N$$

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

Gray-Scott 2D. Two-dimensional Gray-Scott problem [21]:

$$\begin{aligned} u_t &= d_u\nabla^2 u - uv^2 + a(1 - u),\ x, y \in [0, 1], t \in [0, 0.1], \\ v_t &= d_v\nabla^2 v + uv^2 - (a + b)v, \end{aligned}$$

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

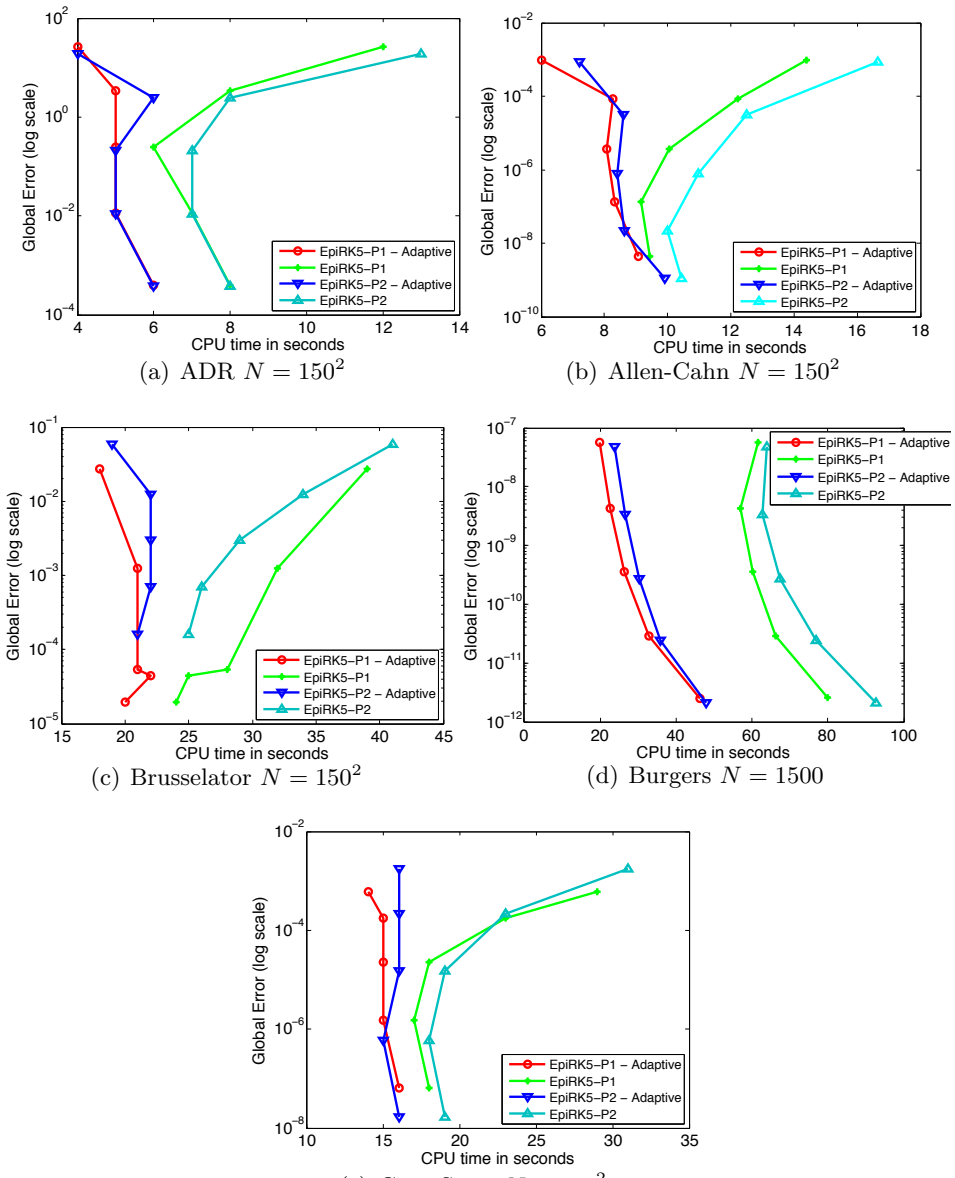
$$\begin{aligned} u &= 1 - e^{-150(x-\frac{1}{2})^2 + (y-\frac{1}{2})^2}, \\ v &= e^{-150(x-\frac{1}{2})^2 + 2(y-\frac{1}{2})^2}. \end{aligned}$$

In all the problems the ∇^2 term was discretized using the standard second-order finite differences. Note that the stiffness in these problems is due primarily to the diffusive term of the equations. These test problems could also be solved using the split exponential schemes which address problems of type $y' = f(y) = Ly + N(y)$ where only the linear term Ly is stiff (e.g. see [37, 28, 31, 11, 16, 43]). While comparative study of split versus unsplit schemes is far outside the scope of this paper, we note that it is far from evident whether split or non-split schemes are more efficient for general, even potentially splittable problems. Unless evaluation of exponential functions of L is particularly optimized for a specific L and/or adapted to be reused over the course of time integration, the unsplit Krylov methods can be as efficient as the split Krylov schemes since both algorithms will require evaluating products $\varphi_k(A)v$ with matrices A having similar spectrums. More detailed discussion of this question can be found in [31, 41, 67].

The precision diagrams shown in figure 4.4 and the statistics of the Krylov algorithm performance in tables 4.4a-4.4e demonstrate the advantages of the adaptive EPIRK methods. As we can see from the figure 4.4 the adaptive integrators are more efficient for all step sizes, with computational savings growing significantly as h increases. Note that the computational savings increase for problems that are more "Krylov-intensive" (i.e. require larger Krylov subspace sizes) such as Burgers, Brusselator or Gray-Scott systems compared to the less "Krylov-intensive" Allen-Cahn equation. The graphs demonstrate that the adaptive integrators do not only have an improved efficiency but also help rectify the precision curve "bending" phenomenon present for non-adaptive schemes for large step sizes (e.g. compare with fig. 4.1). Some oscillatory behavior in the precision graphs of the adaptive schemes (e.g. fig. 4.4a,c,e) indicates that the adaptivity algorithm can be further improved. A better adaptive predictor will not only straighten the "bending" curve but, in fact, reverse the bending (e.g. fig. 4.4d). Such improvement will require better error estimators for the adaptive algorithm and will be the subject of our future investigations.

Tables 5.1a-e provide the statistics of the non-adaptive and adaptive Krylov algorithms averaged over the course of integration. This data gives a more detailed look at the advantages of adaptive methods. As we can see from all the cases, while the total number of Krylov vectors computed each time step is larger for the adaptive schemes, each of the Krylov projections they execute requires a much smaller size of the Krylov space compared to non-adaptive integrators. Thus, the quadratic in Krylov space size complexity of the Arnoldi algorithm ensures that the overall CPU time spent on an adaptive time step is much smaller than for the non-adaptive step. Note that in extreme cases of very small h (e.g. tbl. 4.4c for $h = 0.00625, 0.003125$), the adaptive algorithm can take only one substep, which makes it equivalent to the non-adaptive algorithm. In such cases the efficiency of the adaptive method is slightly worse than for the non-adaptive method since the former requires some additional calculations. However, this difference is essentially negligible compared to the savings achieved if the overall step size is split even once by the adaptive method.

To summarize, the results presented in this section clearly illustrate the advantages of the adaptive exponential integrators over the non-adaptive schemes and suggest avenues for improvement and further development of the new methods. The key advantage of the adaptive Krylov algorithm is that it uses several Krylov projections with *small* Krylov bases as compared to the non-adaptive method which requires one projection but with a *large* Krylov basis. This method is incorporated into the larger framework of EPIRK methods.



(a) ADR $N = 150^2$ (b) Allen-Cahn $N = 150^2$

(c) Brusselator $N = 150^2$ (d) Burgers $N = 1500$

(e) Gray-Scott $N = 150^2$

Figure 4.4: Precision diagrams comparing performance of adaptive versus non-adaptive versions of the EPIRK5-P1 and EPIRK5-P2 integrators for the Advection-Diffusion-Reaction, Allen-Cahn, Brusselator, Burgers and Gray-Scott problems. Note that the axes scale changes from graph to graph.

The EPIRK structure allows derivation of high-order exponential integrators which have the same computational complexity as lower order schemes. Thus the combination of the adaptive Krylov and the EPIRK framework allows us to construct efficient adaptive exponential methods of high-order.

The methodology presented in this paper can be used to derive a multitude of adaptive exponential integrators and investigations of which schemes are the most efficient will be one of the subjects of our future research. A parallel implementation of these schemes is currently underway and will allow testing these integrators on very large scale problems.

Table 4.4a: Average performance statistics of non-adaptive and adaptive Krylov algorithms per Krylov projection within EPIRK5-P1 and EPIRK5-P2 integrators for 2D ADR problem with $N = 150^2$

	Non-adaptive			Adaptive			
	# of Krylov vectors per projection	Total # of Krylov vectors per time step	CPU time	# of substeps per projection	# of Krylov vectors per substep per projection	Total # of Krylov vectors per time step	CPU time
h = 0.04:							
EPIRK5-P1	141.1	424	12.01	6.67	34.2	685	4.04
EPIRK5-P2	149.3	448	13.17	6.83	35.2	724	4.45
h = 0.02:							
EPIRK5-P1	77.6	233	7.84	3.50	39.2	349	4.62
EPIRK5-P2	80.8	242	8.11	1.67	66.7	288	5.83
h = 0.01:							
EPIRK5-P1	44.3	133	6.39	1.00	45.3	136	4.63
EPIRK5-P2	45.8	137	6.54	1.00	46.9	141	4.78
h = 0.005:							
EPIRK5-P1	27.4	82	6.61	1.00	28.1	84	4.78
EPIRK5-P2	28.1	84	6.93	1.00	28.8	68	5.08
h = 0.0025:							
EPIRK5-P1	17.8	54	7.79	1.00	18.1	54	5.86
EPIRK5-P2	18.2	55	8.20	1.00	18.5	56	5.90

Table 4.4b: Average performance statistics of non-adaptive and adaptive Krylov algorithms per Krylov projection within EPIRK5-P1 and EPIRK5-P2 integrators for 2D Allen-Cahn problem with $N = 150^2$

	Non-adaptive			Adaptive			
	# of Krylov vectors per projection	Total # of Krylov vectors per time step	CPU time	# of substeps per projection	# of Krylov vectors per substep per projection	Total # of Krylov vectors per time step	CPU time
h = 0.5:							
EPIRK5-P1	151.7	455	14.4	11.3	32.7	1082	7.03
EPIRK5-P2	156.7	470	16.7	12.5	31.4	1143	7.22
h = 0.25:							
EPIRK5-P1	83.9	252	12.2	8.3	24.5	595	8.26
EPIRK5-P2	87.9	264	12.5	8.4	24.9	622	8.59
h = 0.125:							
EPIRK5-P1	46.8	140	10.0	6.0	18.1	330	8.08
EPIRK5-P2	50.9	150	11.0	6.1	18.4	338	8.41
h = 0.0625:							
EPIRK5-P1	26.0	78	9.16	3.7	14.0	152	8.33
EPIRK5-P2	28.3	85	9.99	3.3	15.1	150	8.62
h = 0.03125:							
EPIRK5-P1	14.4	43	9.43	2.5	10.0	72	9.06
EPIRK5-P2	16.3	49	10.4	2.5	10.6	80	9.90

Table 4.4c: Average performance statistics of non-adaptive and adaptive Krylov algorithms per Krylov projection within EPIRK5-P1 and EPIRK5-P2 integrators for 2D Brusselator problem with $N = 150^2$

	Non-adaptive			Adaptive			
	# of Krylov vectors per projection	Total # of Krylov vectors per time step	CPU time	# of sub-steps per projection	# of Krylov vectors per substep per projection	Total # of Krylov vectors per time step	CPU time
h = 0.05:							
EPIRK5-P1	196.2	589	39.35	9.00	46.6	1281	18.28
EPIRK5-P2	202.8	609	41.46	10.33	45.2	1370	19.48
h = 0.025:							
EPIRK5-P1	122.1	366	32.13	5.92	40.5	719	21.10
EPIRK5-P2	126.4	379	33.55	8.42	32.2	816	21.92
h = 0.0125:							
EPIRK5-P1	75.7	227	27.70	4.00	38.3	386	20.73
EPIRK5-P2	78.9	237	28.92	3.75	36.6	395	22.22
h = 0.00625:							
EPIRK5-P1	46.9	141	25.39	1.00	48.0	144	22.17
EPIRK5-P2	48.8	146	25.97	1.00	50.2	151	21.78
h = 0.003125:							
EPIRK5-P1	28.7	86	24.39	1.00	29.5	88	20.5
EPIRK5-P2	30.0	90	25.04	1.00	30.9	93	20.88

Table 4.4d: Average performance statistics of non-adaptive and adaptive Krylov algorithms per Krylov projection within EPIRK5-P1 and EPIRK5-P2 integrators for 1D Burgers problem with $N = 1500$

	Non-adaptive			Adaptive			
	# of Krylov vectors per projection	Total # of Krylov vectors per time step	CPU time	# of sub-steps per projection	# of Krylov vectors per substep per projection	Total # of Krylov vectors per time step	CPU time
h = 0.02:							
EPIRK5-P1	152.0	456	279.57	17.13	32.0	1626	20.94
EPIRK5-P2	164.7	494	277.28	22.27	29.4	1911	24.75
h = 0.01:							
EPIRK5-P1	88.5	266	114.26	12.31	24.2	905	24.31
EPIRK5-P2	97.7	293	122.96	17.07	21.7	1087	27.52
h = 0.005:							
EPIRK5-P1	51.7	155	82.12	9.18	17.8	501	28.37
EPIRK5-P2	58.1	174	90.97	10.91	17.5	570	31.96
h = 0.0025:							
EPIRK5-P1	30.1	90	77.45	5.07	14.5	234	34.98
EPIRK5-P2	35.0	105	88.84	6.34	14.7	283	37.50
h = 0.00125:							
EPIRK5-P1	18.1	54	86.71	3.64	10.8	128	49.02
EPIRK5-P2	21.4	64	100.56	4.25	11.7	151	51.21

Table 4.4e: Average performance statistics of non-adaptive and adaptive Krylov algorithms per Krylov projection within EPIRK5-P1 and EPIRK5-P2 integrators for 2D Gray-Scott problem with $N = 150^2$

	Non-adaptive			Adaptive			
	# of Krylov vectors per projection	Total # of Krylov vectors per time step	CPU time	# of sub-steps per projection	# of Krylov vectors per substep per projection	Total # of Krylov vectors per time step	CPU time
h = 0.05:							
EPIRK5-P1	167.2	502	28.65	12.33	33.3	1164	14.17
EPIRK5-P2	173.0	519	31.43	9.17	40.8	1097	16.49
h = 0.025:							
EPIRK5-P1	99.5	299	22.76	7.33	28.7	641	15.18
EPIRK5-P2	104.2	313	23.39	9.75	24.5	717	15.61
h = 0.0125:							
EPIRK5-P1	59.6	179	18.08	4.04	25.1	307	14.92
EPIRK5-P2	63.0	189	19.13	4.46	25.6	329	16.22
h = 0.00625:							
EPIRK5-P1	35.9	108	16.52	1.69	28.6	127	14.70
EPIRK5-P2	38.0	114	18.13	1.67	32.8	131	15.39
h = 0.003125:							
EPIRK5-P1	21.7	65	18.08	1.40	19.1	74	15.55
EPIRK5-P2	23.1	69	18.62	1.33	20.8	77	16.11

4.7 Acknowledgements

This material is based upon work supported by the National Science Foundation, Computational Mathematics Program, under Grant No. 1115978. The authors would like to thank Will Wright for helpful discussions and for sharing his code that helped us better understand his algorithm.

5 Implementation of parallel adaptive-Krylov exponential solvers for large scale stiff problems

5.1 Abstract

Recently exponential integrators have been receiving increased attention as a means to solve large stiff systems of ODEs. Preliminary performance analysis demonstrated that exponential integrators hold promise compared to state-of-the-art implicit methods. However much work remains to be done to understand in detail possible computational advantages these methods may offer in practice. This is particularly true for supercomputer-scale problems as there has been very little work on parallelizing exponential methods. In this paper we describe an implementation of a suite of parallel exponential solvers. We present some performance tests on four stiff benchmark problems of a particular adaptive-Krylov exponential propagation iterative Runge-Kutta (EPIRK) method from the suite, and compare efficiency with the Newton-Krylov implicit solver CVODE.

5.2 Introduction and background

Exponential integrators have received renewed interest in recent years as a means to solve large stiff systems of ODEs. First appearing in the literature in the 1960's [12, 55, 37], they were originally limited to the treatment of small systems, due to the high computational cost of evaluating the exponential-like functions of a matrix. A proposal by Van der Vorst [15] for using Krylov projection techniques for the efficient evaluation of the matrix exponential terms made application of exponential integrators to large-scale systems feasible. Since then, attention to exponential integrators has increased and a number of methods for systems of ODEs have been proposed [19, 5, 20, 27, 13, 33, 35, 30, 66, 53, 32, 72].

Preliminary results [41] show that exponential integrators can be competitive with other classes of integrators at the single processor scale, most notably compared with Newton-Krylov implicit methods, which are the currently most widely used class of methods for large scale stiff problems [34]. However more research is needed to understand what, if any, computational advantages exponential methods offer for practical applications. In particular, performance of exponential methods on parallel systems remains largely untested. To our knowledge there has been only one study of a parallel implementation of exponential integrators [42], using Leja point approximation instead of Krylov projection, but comparisons with other integrators were not made.

In this paper we describe a parallel implementation of a suite of exponential solvers designed for easy extensibility to both new schemes and different techniques for approximation of the matrix exponential terms. We detail the structure and Krylov-based implementation of a particular member of the suite, the EPIRK5P1 integrator taken from the class of exponential propagation iterative Runge-Kutta (EPIRK) methods [67], and present performance results using four stiff benchmark problems. The method was designed in [69]

to efficiently take advantage of an adaptive Krylov technique [50], making it particularly well-suited for dealing with the wide spectra associated with parallel-scale stiff problems. As a performance baseline, we compare the efficiency of EPIRK5P1 against the CVODE solver [25], a parallel implementation of a Newton-Krylov implicit solver used today on state-of-the-art massively parallel systems to treat a wide variety of large-scale problems of current scientific interest. While Newton-Krylov solvers use preconditioning when possible to maximize efficiency and scalability, currently no efficient preconditioners exist for Krylov-based exponential integrators and the issue of preconditioning is not addressed in this work.

The structure of this paper is as follows. Section 5.3 describes EPIRK5P1 and its adaptive Krylov implementation and contrasts its mathematical structure with the Newton-Krylov BDF methods found in CVODE. The software structure of the parallel exponential solver suite is also detailed. Section 5.4 details the stiff test problems and the setup of the experiments. The results of the numerical experiments and a discussion of the performance difference between the integrators is presented in section 5.5. Finally, some conclusions and possibilities for future work are given in section 5.6.

5.3 Description of EPIRK5P1

We are considering exponential time integrators for problems of the form

$$y' = f(y(t)), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^N, \tag{5.1}$$

where N is large and the system is stiff. To derive an exponential method, the system is first linearized using Taylor expansion around y_0 to give

$$y' = f(y_0) + J_0(y - y_0) + r(y) \tag{5.2}$$

where $J_0 = f'(y_0)$ is the Jacobian matrix, and $r(y) = f(y) - f(y_0) - f'(y_0)(y - y_0)$ is the nonlinear remainder of the expansion. Then applying the integrating factor $e^{-J_0 t}$ and performing a change of integration variable, the integral form of the system

$$y(t_0 + h) = y(t) + (e^{hJ_0} - I)(hJ_0)^{-1} h f(y) + h \int_0^1 e^{hJ_0(1-s)} r(y(t_0 + hs)) ds \tag{5.3}$$

is produced. An exponential method is constructed by numerically approximating the integral term. In particular, approximating the $r(y(t_0+hs))$ term inside the integral using polynomial approximation will result in linear combinations of the exponential-like φ-functions

$$\varphi_k(z) = \int_0^1 e^{z(1-\theta)} \frac{\theta^{k-1}}{(k-1)!} d\theta, \quad k = 0, 1, 2, \cdots$$

acting on vectors $b_i \in \mathbb{R}^N$, i.e. expressions of the form

$$\varphi_0(hJ)b_0 + \varphi_1(hJ)b_1 + \varphi_2(hJ)b_2 + \cdots + \varphi_i(hJ)b_i. \tag{5.4}$$

A particular type of quadrature produces the exponential propagation iterative Runge-Kutta (EPIRK) class of methods, described in detail in [67]. The numerical experiments of sections 5.4 and 5.5 will focus on one EPIRK method chosen from the software suite, the EPIRK5P1 scheme [69] shown in formula (5.5) with coefficients listed in Table 5.1.

$$
\begin{aligned}
Y_1 &= y_0 + a_{11}\varphi_1(g_{11}hJ_0)hF_0 \\
Y_2 &= y_0 + a_{21}\varphi_1(g_{21}hJ_0)hF_0 + a_{22}\varphi_1(g_{22}hJ_0)hr(Y_1) \\
y_1 &= y_0 + b_1\varphi_1(g_{31}hJ_0)hF_0 + b_2\varphi_1(g_{32}hJ_0)hr(Y_1) + b_3\varphi_3(g_{33}hJ_0)h[-2r(Y_1) + r(Y_2)],
\end{aligned}
\tag{5.5}
$$

EpiRK5P1:

$$
\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 0.3512959269505819 & & \\ 0.8440547201165712 & 1.6905891609568963 & \\ 1.0 & 1.27271273173568923 & 2.271459926542262227 \end{bmatrix}
$$

$$
\begin{bmatrix} g_{11} & & \\ g_{21} & g_{22} & \\ g_{31} & g_{32} & g_{33} \end{bmatrix} = \begin{bmatrix} 0.3512959269505819 & & \\ 0.8440547201165712 & 1.0 & \\ 1.0 & 0.71111095364366870 & 0.62378111953371494 \end{bmatrix}
$$

Table 5.1: Coefficients of EpiRK5P1.

We choose this particular method for its fifth order accuracy and because it was designed specifically to take advantage of the adaptive Krylov technique described in section 5.3.2, making it particularly efficient.

Approximation of the $\varphi_k(A)v$ terms which make up the methods, e.g. the term $\varphi_1(g_{11}hJ_0)hF_0$ in EPIRK5P1, is the primary computational challenge for exponential integrators. Today there are multiple techniques which are potentially suitable for large-scale problems, such as Krylov projection methods, polynomial approximation, contour integrations, and others. It remains to be seen which methods are most efficient for which types of problems. A preliminary comparison of the efficiency of some of the methods on various problems can be in found in [8]. In this paper we focus on Krylov projection techniques due to their applicability to a wide range of problems. In sub-section 5.3.1 we describe the basic application of Krylov projection techniques to the evaluation of the $\varphi_k(A)v$ terms, and in sub-section 5.3.2 we detail the particular adaptive Krylov algorithm used in the implementation of EPIRK5P1.

5.3.1 Krylov approximation of the $\varphi_k(hJ)v$ terms

As mentioned in the introduction, traditional techniques for evaluating the $\varphi_k(hJ)v$ terms, such as Taylor or Padé approximation, are prohibitively expensive for large matrices, so we turn to Krylov subspace projection techniques. The terms are approximated by projecting onto the Krylov subspace $K_m(hJ, v) = \text{span}\{v, (hJ)v, (hJ)^2v, \ldots, (hJ)^{m-1}v\}$. An orthonormalized form of the Krylov basis is generated iteratively using the Arnoldi iteration [57], which uses a modified Gram-Schmidt process to orthonormalize the basis vectors and store them column-wise in a matrix V_m. An $m \times m$ upper Hessenberg matrix

$$
H_m = V_m^T(hJ)V_m \tag{5.6}
$$

is also computed as a side product. Since V_m is an orthogonal matrix, the product of a function of a matrix times a vector $f(hJ)v$ can be approximated by orthogonally projecting

onto the Krylov subspace as

$$f(hJ)v \approx V_m V_m^T f(hJ) V_m V_m^T v.$$

By equation (5.6),

$$V_m^T f(hJ) V_m \approx f(H_m)$$

allowing $f(hJ)v$ to be approximated by

$$f(hJ)v \approx V_m f(H_m) V_m^T v.$$

Further, since $v/||v||_2$ is the first column of V_m, the approximation simplifies to

$$f(hJ)v \approx ||v||_2 V_m f(H_m) e_1, \tag{5.7}$$

which is the approximation used in implementations. This approximation has lower computational cost than direct evaluation because H_m is expected to be a small matrix, making computation of $f(H_m)$ considerably cheaper than evaluation of $f(hJ)$. $f(H_m)$ is typically computed using Taylor or Padé approximation [46], such as with the Padé algorithm of Higham [24]. Benchmarks show that the cost of computing $f(H_m)$ is typically negligible compared to the cost of producing the Krylov basis itself.

We note that Arnoldi algorithm does not need an explicit representation of the J matrix and the generation of the Krylov basis requires only the implementation of a routine which evaluates the action of the matrix on a vector, i.e. a procedure $JTimesV(v)$ which returns a vector equivalent to $J * v$. This so-called "matrix-free" implementation is typical in Krylov-based methods [34] and is employed in both EPIRK5P1 and CVODE.

The Arnoldi algorithm can be applied to each $\varphi_k(hJ)v$ term in an exponential integrator individually in order to compute the next time step. For example in EPIRK5P1 shown in (5.5), there are six $\varphi_k(hJ)v$ terms which can each be computed with a separate invocation of the Arnoldi algorithm. However, the Arnoldi iteration has a scale invariance property which can be used to compute some of the terms together with a single Krylov basis. If by equation (5.6), $H_m = V_m^T(hJ)V_m$ for matrix hJ, then $\alpha H_m = V_m T(\alpha hJ)V_m$ for matrix αhJ. In other words, terms of the form $\varphi_k(\alpha hJ)v$ which involve the same vector v can be computed using a single basis even if the Jacobian matrix is scaled differently in each term. In the case of EPIRK5P1, this means the three terms $\varphi_1(g_{11}hJ_0)hF_0$, $\varphi_1(g_{21}hJ_0)hF_0$, and $\varphi_1(g_{31}hJ_0)hF_0$ can be computed using a single Krylov basis, the two terms $\varphi_1(g_{22}hJ_0)hr(Y_1)$ and $\varphi_1(g_{32}hJ_0)hr(Y_1)$ can be computed using a second basis, and the term $\varphi_3(g_{33}hJ_0)h[-2r(Y_1)+r(Y_2)]$ with a third basis, for a total of three invocations of the Arnoldi algorithm to compute each time step. This forms an important contrast with Newton-Krylov implicit methods which must perform the Krylov algorithm each Newton iteration and thus a variable number of times each time step. This point will be discussed further in section 5.3.3.

While evaluating the $\varphi_k(H_m)$ terms using Krylov approximation is significantly more efficient than direct methods such as Taylor or Padé approximation, performing the Krylov iterations is still the primary computational cost of Krylov-based integrators. The cost is determined by how rapidly the iterations reach a desired accuracy, i.e. how many iterations m are needed. The factors which determine this rate of convergence are the function f, the spectrum of J, the magnitude of h, and the magnitude and orientation of v. It was shown in [26] that in the special case where J is a Hermitian negative semi-definite matrix, the rate of

	Number of Krylov vectors		
h	$\varphi_1(hJ_0)f(u_0)$	$\varphi_2(hJ_0)f(u_0)$	$\varphi_3(hJ_0)f(u_0)$
0.01	62	56	49
0.005	40	35	31
0.0025	26	22	19

Table 5.2: Effect of scaling of the Jacobian by the time step size h on Krylov basis size

convergence is superlinear once $m \geq \sqrt{||hJ||}$ (although in practice it usually occurs at much smaller m), but theoretical convergence results are difficult in general. Theoretical results [26] and numerical experiments [66, 41] indicate that the Krylov iteration converges faster for $f = \varphi_k(z)$ as in exponential integrators compared to when it is applied to a rational function $1/(1-z)$ as in Newton-Krylov implicit methods.

The degree of scaling of the Jacobian by the time step h is particularly important, both because its impact on the number of required Krylov vectors needed for a given accuracy is high and because the step size can be controlled, offering a means to modulate the computational cost. We illustrate the effect of the scaling on the Krylov convergence with the following experiment. Consider the 2D Gray-Scott problem

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

with $d_u = 0.2$, $d_v = 0.1$, $a = 0.04$, and $b = 0.06$, periodic boundary conditions, and initial conditions

$$\begin{aligned} u &= 1 - e^{-150(x-\frac{1}{2})^2 + (y-\frac{1}{2})^2}, \\ v &= e^{-150(x-\frac{1}{2})^2 + 2(y-\frac{1}{2})^2}. \end{aligned}$$

The problem is discretized using the second order centered finite difference approximation on a 150×150 uniform grid. Table 5.2 displays the number of Krylov vectors needed to achieve a tolerance of 10^{-6} for the terms $\varphi_1(hJ_0)v$, $\varphi_2(hJ_0)v$, and $\varphi_3(hJ_0)v$ when $J_0 = f'(y_0)$ and $v = f(y_0)$, for three different values of h. We can see that each reduction in the magnitude of h by $1/2$ significantly reduces the number of Krylov vectors needed to achieve the desired accuracy. The computational cost of performing the Krylov iteration grows quadratically with the number of Krylov vectors needed, i.e. as $O(m^2)$, so the scaling of the Jacobian by h has a marked effect on the computational cost of Krylov-based integrators.

The quadratic growth in cost with basis size makes controlling Krylov cost through scaling of the matrix crucial in Krylov-based integrators due to the following phenomenon. Ordinarily increasing step size h in a time integrator results in lower computational cost because fewer time steps are needed to reach the final solution. However in a Krylov-based integrator, if h becomes too large then the increased Krylov cost may outweigh the savings from computing fewer time steps. This phenomenon is portrayed by the solid line labeled "No Adaptivity" in the illustration (not actual data) of a precision diagram, shown in Figure 5.1. Each point on the curve corresponds with a solution computed at a particular time step size. As represented by the lower right point, for the smallest step size a high accuracy solution of 10^{-8} is computed but at high computational cost as many time steps are needed.

As the step size is increased, less accurate solutions are produced but initially at lower computational cost, such as for the points at accuracies 10^{-7} and 10^{-6}. However as the step size is increased further, even less accurate solutions are produced but now at increasing computational cost, such as for the points with accuracies 10^{-5} and 10^{-4}. Clearly even if only a solution of accuracy 10^{-4} or 10^{-5} is needed, it is better to use a smaller time step and compute a solution of accuracy 10^{-6} at lower cost. We would like an algorithm which is able to adaptively scale the matrix to give the lowest cost solution for the accuracy we need. The hypothetical performance of such an algorithm is portrayed by the dashed line labeled "With Adaptivity". The points on the vertical portion of the curve above the inflection point with accuracies 10^{-5} and 10^{-4} represent solutions computed more efficiently if we were to modulate the step size to minimize cost. For example if our accuracy requirement for a solution is 10^{-4}, rather than computing the solution to an accuracy of 10^{-4} requiring 300 seconds of CPU time as on the solid line, we could instead lower the step size to that used for the solution with accuracy 10^{-6} and get a more accurate solution for only 10 seconds of time. The points below the inflection of the curve with accuracies 10^{-6}, 10^{-7}, and 10^{-8} are the same as on the solid line, and represent solutions for which lowering the time step size further would increase the computational cost instead of lowering it. Since increasing the step size would give too inaccurate a solution, the step size is already at its optimal value and the adaptivity algorithm cannot help further.

As described, modulation of h is one means to control the Krylov cost. However, the downside of controlling cost through the step size is that h scales the Jacobian for all the $\varphi_k(hJ_0)v$ terms in a scheme together without giving individual control of each term. In the next section we describe a more sophisticated adaptive Krylov algorithm which allows each term to be modulated independently, and describe its use in EPIRK5P1.

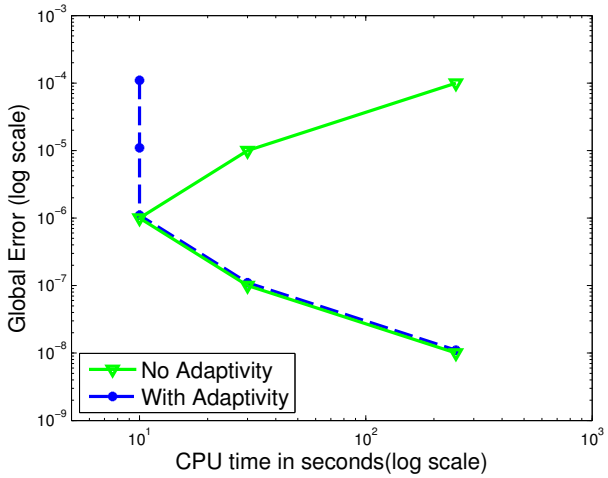


Figure 5.1: Illustration of the importance of Krylov adaptivity for the control of computational cost.

5.3.2 Krylov adaptivity

The approach to Krylov adaptivity we implemented was proposed in [62, 50] to evaluate linear combinations of the form

$$\varphi_0(A)v_0 + \varphi_1(A)v_1 + \cdots + \varphi_p(A)v_p, \tag{5.8}$$

more efficiently. It's based on the observation that

$$u(t_{k+1}) = \varphi_0(\tau_k A)u(t_k) + \sum_{i=1}^{p} \tau_k^i \varphi_i(\tau_k A) \sum_{j=1}^{p-i} \frac{t_k^j}{j!} v_{i+j}, \quad \tau_k = t_{k+1} - t_k, \tag{5.9}$$

is the solution to the initial value problem

$$u'(t) = Au(t) + v_1 + tv_2 + ... + \frac{t^{p-1}}{(p-1)!}v_p, \quad u(0) = v_0. \tag{5.10}$$

By stepping from $t_0 = 0$ to $t_1 = t$, equation (5.9) implies that

$$u(t) = \varphi_0(tA)v_0 + t\varphi_1(tA)v_1 + t^2\varphi_2(tA)v_2 + ... + t^p\varphi_p(tA)v_p, \tag{5.11}$$

and so linear combination (5.8) can be interpreted as the solution to the ODE (5.10) at $t = 1$. Using the recurrence $\varphi_q(A) = \varphi_{q+1}(A)A + \frac{1}{q!}I$, equation (5.9) can be simplified to

$$u(t_{k+1}) = \tau_k^p \varphi_p(\tau_k A)w_p + \sum_{j=0}^{p-1} \frac{\tau_k^j}{j!} w_j, \tag{5.12}$$

where the w_j vectors be computed recursively by

$$w_0 = u(t_k), \quad w_j = Aw_{j-1} + \sum_{l=0}^{p-j} \frac{t_k^l}{l!} v_{j+l}, \quad j = 1, ..., p. \tag{5.13}$$

Equation (5.12) provides a way to compute expressions of type (5.8) adaptively by stepping the solution $u(t)$ from $t = 0$ to $t = 1$ along time points $0 = t_0 < t_1 < \cdots < t_k < t_{k+1} = t_k + \tau_k < \cdots < t_K = t_{end} = 1$. For each $u(t_{k+1})$, a single Krylov projection must be performed for the $\tau_k^p \varphi_p(\tau_k A)w_p$ term. Therefore over the K time steps from $t_0 = 0$ to $t_K = t_{end} = 1$, a total of K Krylov projections must be computed. Since $0 < \tau_k < 1$, each projection will cost fewer Krylov vectors than computing $\varphi_p(A)w_p$ in a single projection with an unscaled matrix. Surely the more substeps taken, the cheaper will be each projection. The key is to find the best tradeoff between the number of Krylov vectors needed per projection versus the K number of projections. The implementation of EPIRK5P1 uses the approach described in [50] for choosing τ_k. It uses an error estimator to predict how many Krylov vectors will be needed for a particular size of τ_k, and then chooses the τ_k anticipated to meet the error tolerance for the lowest number of flops.

In section 5.3.1 we noted that terms $f(A)v$ and $f(cA)v$ sharing the same v vector can be computed together using a single Krylov evaluation. For example, in EPIRK5P1 the three terms $\varphi_1(g_{11}jJ_0)hF_0$, $\varphi_1(g_{21}jJ_0)hF_0$, and $\varphi_1(g_{31}jJ_0)hF_0$ can all be computed together with the same Krylov basis, allowing all three terms to be evaluated for the cost of one

projection. That work saving property can also be used with formula (5.12) as well. We explain by example. From formula (5.11), we see that if we wish to compute a linear combination composed of a single term $\varphi_k(tA)v_k$, i.e. if all other terms $\varphi_j(tA)v_j$ where $j \neq k$ are zero, then we can compute it as $\varphi_k(tA)v_k = u(t)/t^k$. Therefore to compute the term $\varphi_1(g_{11}hJ_0)hF_0$ in EPIRK5P1, we can compute it as $u(g_{11})/g_{11}$. We can do similarly for the $\varphi_1(g_{21}hJ_0)hF_0$ and $\varphi_1(g_{31}hJ_0)hF_0$ terms by computing $u(g_{21})/g_{21}$ and $u(g_{31})/g_{31}$ respectively. Clearly then all three terms can be evaluated by stepping formula (5.12) over times $t_0 = 0$ to $t_K = t_{end} = 1$ as long as g_{11}, g_{21}, and g_{31} are included in the set of times $\{t_k\}_{k=0}^K$. We emphasize that this approach is possible only if the terms to be grouped are composed of single φ_k terms and not linear combinations. (Note, though, that the $\varphi_3(g_{33}hJ_0)h[-2r(Y_1) + r(Y_2)]$ term in EPIRK5P1 does not share its vector with any other terms and therefore is allowed to be composed of any linear combination of $\varphi_k(z)$ functions.) The scheme of EPIRK5P1 was derived explicitly with this property in mind [69].

Ideally the τ_k's are chosen to minimize the cost by optimizing the number of Krylov iterations per projection versus the number of projections K. Choosing τ_k such that the times t_k include intermediate times $\hat{t}_j$, e.g. $0 = t_0 < t_1 < \cdots < \boldsymbol{\hat{t}_j = g_{11}} < \cdots < t_k < t_{k+1} = t_k + \tau_k < \cdots < t_{end} = 1$, might result in sizes for τ_k which give suboptimal cost. In fact it's possible to compute intermediate solutions $u(\hat{t}_j)$ without computing additional Krylov basis other than those for times t_k chosen without regard to $\hat{t}_j$. If for example $t_k < \hat{t}_j < t_{k+1} = t_k + \tau_k$, besides computing $u(t_{k+1})$ using formula (5.12) with τ_k, we can also compute $u(\hat{t}_j)$ using the same formula, but instead using $\hat{\tau}_j = \hat{t}_j - t_k$ instead of τ_k. By the scaling property of the Arnoldi iteration, the same Krylov basis can be used for both $u(t_{k+1})$ and $u(\hat{t}_j)$. Since $\hat{\tau}_j < \tau_k$, the solution for $u(\hat{t}_j)$ will be of equal or higher accuracy than that for $u(t_{k+1)}$.

To illustrate the computational benefit of using Krylov adaptivity, the precision diagram in Figure 5.2 shows two implementations of EPIRK5P1 applied to the Gray-Scott 2d problem from section 5.3.1. The green curve corresponds to a non-adaptive Krylov implementation, and the blue curve to an adaptive Krylov implementation. Both implementations use constant time stepping. Each point on the curves gives the CPU time and error for the solution computed with a particular time step size, starting with a step size of 0.1 and each subsequent point half the size of the previous. As can be seen, the adaptive Krylov implementation consumes significantly less CPU time than the non-adaptive implementation, particularly for large step sizes when the Jacobian matrix is scaled coarsely.

5.3.3 Comparison with CVODE

We compare the performance of EPIRK5P1 with the parallel Newton-Krylov implicit ODE solver CVODE [25], a variable-order, variable-timestep implementation of the fixed-leading coefficient variant of the BDF schemes

$$\sum_{i=0}^{q} \alpha_{n,i} y^{n-i} + h_n \beta_{n,0} f(y^n) = 0. \tag{5.14}$$

The y^n are the approximations to $y(t_n)$, q is the order of the method, which ranges from one to five, the coefficient $a_{n,0} = -1$, and the remaining coefficients $a_{n,i}$ and $\beta_{n,0}$ are determined by the order and the previous step sizes.

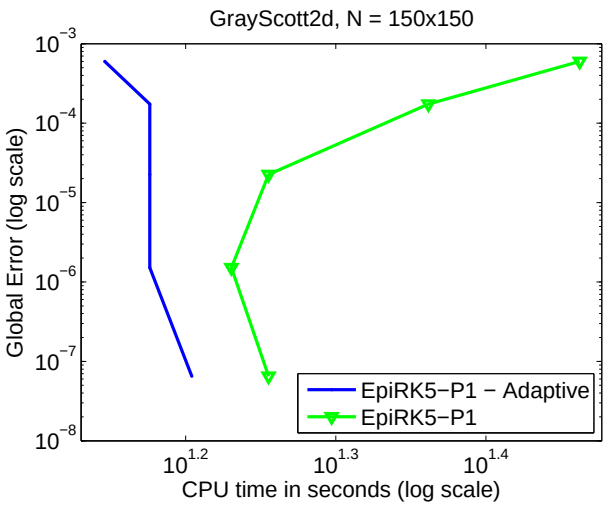


Figure 5.2: Adaptive Krylov versus non-adaptive Krylov implementations of EPIRK5P1.

To compute the implicit term y^n, each integration step the nonlinear system

$$G(y^n) = y^n - h_n \beta_{n,0} f(y^n) - \sum_{i=1}^{q} \alpha_{n,i} y^{n-i} = 0, \tag{5.15}$$

is solved using the Newton iteration. Let $y^{n(k)}$ be the k-th iteration of the Newton iteration. The initial guess of the Newton iteration $y^{n(0)}$ is a predicted value computed explicitly from previous time steps. This gives an initial guess which is quite accurate and allows the Newton iteration to converge in a small number of iterations.

Each step of the Newton iteration the solution to the linear system

$$M[y^{n(m+1)} - y^{n(m)}] = -G(y^{n(m)}), \tag{5.16}$$

must be approximated, where M is the matrix $I - h_n \beta_{n,0} J_n$. The approximation is done using the GMRES Krylov solver [58]. A major determiner of the difference in efficiency between EPIRK5P1 and CVODE is the rate of convergence of the Krylov iteration when computing $\varphi_k(g_{ij} h J)v$ versus the linear system (5.16) by GMRES. It was shown in [26] that for Hermitian negative semi-definite matrices, the rate of convergence for computing $\varphi_k(A)v$ is faster than for approximating the linear system $[I - A]x = v$, when the matrix is the same in both cases. Theoretical results are difficult for general matrices, but the idea has been numerically demonstrated for some other types of matrices as well, e.g. in [66, 41]. However, as seen in section 5.3.1 the scaling of the matrix has a significant impact on the number of Krylov iterations as well. For EPIRK5P1, the Jacobian is scaled by $h g_{ij}$. For CVODE, the Jacobian is scaled by $h \beta_{n,0}$. Clearly then the scaling of the Jacobian will be influenced by what step sizes h must be chosen to achieve a specified error tolerance, and by the particular coefficients of the scheme, both of which can be quite different between the two integrators.

In EPIRK5P1, terms $\varphi_k(g_{ij} h J)v$ with the same vector v are computed together each time step using the same Krylov basis. The scaling of the matrix used in the evaluation is

determined by the largest g_{ij} coefficient for the group, which corresponds to the bottom row of Table 5.1. We see that the Jacobian of the first Krylov evaluation is scaled by $1.0h$, the second evaluation by about $0.71h$, and the third by about $0.62h$. In CVODE the Jacobian is scaled by $h\beta_{n,0}$, where $\beta_{n,0}$ depends upon the order of the BDF scheme. While CVODE changes the order dynamically each time step, almost all time steps are integrated using the fourth or fifth order scheme on the test problems. In the fourth order case $\beta_{n,0}$ is $11/25$ and in the fifth order case it is $60/137$, coefficients significantly smaller than the g_{ij} coefficients of EPIRK5P1.

How these factors balance out on the test problems will be discussed in the results section.

5.3.4 The software

EPIRK5P1 is implemented as part of a suite of exponential time integrators written in C++ and parallelized using MPI. Since the software is oriented towards developing and testing new exponential methods and new approximation methodologies for the $\varphi_k(A)v$ terms, it is designed with easy extensibility in mind. New schemes within a previously implemented class, e.g. the EPIRK class, can be created simply by specifying their coefficients, and new approximation algorithms for the $\varphi_k(A)v$ terms can be added without affecting the implementation of previously created methods. We discuss the software structure further below. The software is also designed to accept problems written for the CVODE solver suite. This means the exponential methods can accept as input parameters handles to the routines implementing the right-hand-side $f(v)$ and Jacobian times a vector $J * v$ functions from a problem written for CVODE. Time integrating a problem written for CVODE with an exponential integrator can thus be done in just a few lines of initialization code.

The other configuration parameters which can be specified are the following :

- Initial time step size
- Maximum time step size
- Absolute error tolerance
- Relative error tolerance
- Initial integration time
- Final integration time
- $\varphi_k(A)v$ approximation algorithm, e.g. adaptive-Krylov
- Maximum Krylov basis size (when using Krylov approximation to evaluate the $\varphi_k(hJ)v$ terms)

The basic structure of the software is illustrated in Figure 5.3. Each exponential integrator belongs to a corresponding mathematical class, e.g. EPIRK5P1 belongs to the EPIRK class of methods. The mathematical structure of each class is implemented using NVECTOR in a C++ abstract base class, represented in the diagram by the dashed box labeled "Abstract class implementations". A new scheme within the class is derived by inheriting from the abstract base class and specifying its coefficients. In the figure the specific

schemes are the circles within the dashed box labeled "Schemes", with each scheme connected to its parent base class by a line. The use of any particular approximation algorithm for the $\varphi_k(A)v$ terms, e.g. Krylov approximation, is not hardcoded into any of the class implementations, but rather evaluation of the terms is done via an abstract interface, represented by the rectangle with rounded corners in the middle of the diagram. This allows the exponential integrators to use different approximation algorithms simply by changing an input configuration parameter. The abstract interface accepts requests for approximations to a set of terms of the form $\varphi_k(A)v$ and hands the requests to the specified underlying approximation algorithm, as represented by the lines connecting the abstract interface to the approximation algorithms inside the dashed rectangle labeled "$\varphi_k(A)v$ approximation". The approximation routine is then free to internally compute the terms in whatever manner will give the best efficiency. The results are always returned through the interface in an NVECTOR structure. To maximize efficiency, it is important to call the approximation algorithm such that all terms which can share computational cost be listed together. For example, the current implementation for the EPIRK class calls the interface three times per time step for a three stage method like EPIRK5P1. On the first call it requests the interface to return approximations to the three terms $\varphi_1(g_{11}hJ_0)hF_0$, $\varphi_1(g_{21}hJ_0)hF_0$, and $\varphi_1(g_{31}hJ_0)hF_0$. If the interface passes that request to the non-adaptive Krylov approximation algorithm, the routine would compute the three terms internally using a single Krylov basis. On the second call it requests approximations to the two terms $\varphi_1(g_{22}hJ_0)hr(Y_1)$ and $\varphi_1(g_{32}hJ_0)hr(Y_1)$, both of which would be computed together using a second basis, and the third time for the term $\varphi_3(g_{33}hJ_0)h[-2r(Y_1) + r(Y_2)]$ which is computed alone. Of course not all approximation methods operate at their best efficiency for the same grouping of terms. For example as discussed in section 5.3.2, the adaptive-Krylov approximation algorithm can compute together all terms sharing the same v vector, sharing the cost amongst all the terms. However, iteration (5.12) can also be used to compute together all terms in the same Runge-Kutta stage, e.g. it can compute together the three terms $\varphi_1(g_{31}hJ_0)hF_0$, $\varphi_1(g_{32}hJ_0)hr(Y_1)$, $\varphi_1(g_{31}hJ_0)hF_0$ in the third stage, sharing the cost amongst all three terms. For some problems that may be more efficient. Therefore an implementation within one of the abstract base classes will always give a correct result no matter what approximation algorithm is specified, but might not give the best efficiency if the grouping of terms it specifies is non-ideal for the particular algorithm. When using a different approximation technique, it might be better to re-implement the class using a grouping of terms that is most efficient for it. Unfortunately there seems to be no simple interface which gives both correctness and maximum efficiency in all cases.

Parallelization of the exponential methods is done at the vector level in the same manner as in CVODE, i.e. through CVODE's parallelized NVECTOR data structure. All vectors being computed, whether the solution at the current time step $y(t_n)$ or intermediate scratch vectors, are internally represented as NVECTOR vectors. The usual set of algebraic vector operations, such as vector addition and scaling, norms, etc. can be applied to the NVECTOR vectors, and those operations are done in parallel. Parallelization is done in the standard data-parallel approach in which each N-dimensional vector is split across all the processors and MPI communication is used to implement the vector algebraic operations. This has the advantage that the commonly used operations of vector addition and scaling are computed locally without any inter-processor communication. By defining the time integrator's scheme in terms of the NVECTOR operations, the method is naturally

parallelized. The vectors outputs from the right-hand-side $f(v)$ and Jacobian times vector $J * v$ functions, which constitute implementations of ODE problems for CVODE, are also represented as NVECTOR data. Choosing to use the NVECTOR structure internally in the exponential methods makes compatibility with those functions simple. It also removes underlying implementational differences in the vector operations as a source of performance difference when comparing the efficiency of exponential solvers with CVODE, allowing a more direct comparison of the efficiency of the algorithms themselves. The parallelization of the $f(v)$ and $J * v$ routines are handled by the problem implementation and in general must be done directly using MPI. Discretization of the differential operators typically requires exchange of boundary values between processors and other operations which don't map onto the standard NVECTOR operations. Naturally the final computed result of the $f(v)$ and $J * v$ functions is still returned in an NVECTOR data structure.

The primary computational and communication costs in CVODE and the exponential integrators are the vector dot products used in the Krylov iteration, and the evaluation of $f(y)$, and the $J * v$ routine, all of which grow in communication cost with increasing size of N, and thus being a scalability constraint. Nevertheless, the scalability bottleneck of those operations is greatly overshadowed by the algorithmic scalability constraints of the Krylov iteration. As problem size increases, the spectrum of the Jacobian matrix widens and the number of Krylov iterations needed to achieve a given accuracy tolerance grows. We saw this phenomenon in the form of scaling of the Jacobian matrix in section 5.3.1, as illustrated in Table 5.2. This bottleneck can be treated with preconditioning or with Krylov adaptivity. We discuss how Krylov adaptivity ameliorates the growth in Krylov cost in section 5.5.

It should also be noted that the maximum Krylov basis size and the maximum time step size can have a significant effect on efficiency. A maximum basis size is typically needed for large problems due to limited computer memory, but judiciously constraining the basis size has the effect of forcing the integrator to take correspondingly smaller time steps to maintain accuracy. This can prevent the CPU time from moving past the inflection in Figure 5.1, as per the arguments in section 5.3.1, and acts in a manner similar to Krylov adaptivity. Clearly a shrewdly chosen maximum time step size can also act similarly. Configuration of the parameters for greatest efficiency is problem dependent, and we do not attempt to optimize the parameters in the numerical experiments, but we give a simple example of the phenomenon in the numerical results section 5.5.

5.4 Setup of experiments

We tested the performance of EPIRK5P1 on the following four stiff test problems.

ADR 2D. Two-dimensional advection-diffusion-reaction equation [9]:

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

with $\epsilon = 1/100$, $\alpha = -10$, $\gamma = 100$, homogenous Neumann boundary conditions, and initial conditions $u_0 = 256(xy(1 - x)(1 - y))^2 + 0.3$.

Allen-Cahn 2D. Two-dimensional Allen-Cahn equation [4]:

$$u_t = \alpha \nabla^2 u + u - u^3, \quad x, y \in [-1, 1], t \in [0, 1.0]$$

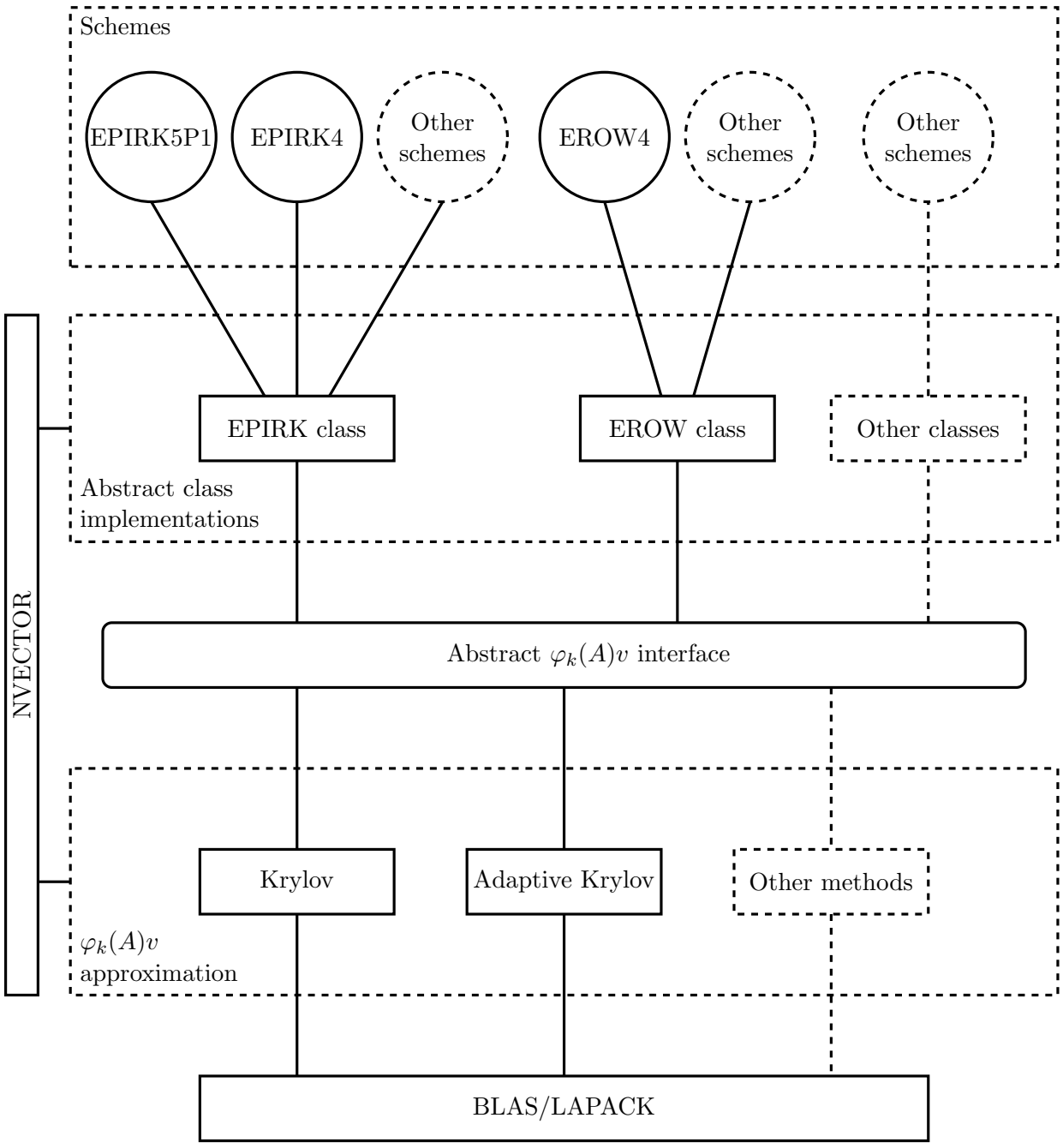


Figure 5.3: Software structure

with $\alpha = 0.1$, no-flow boundary conditions, and initial $u = 0.1 + 0.1\cos(2\pi x)\cos(2\pi y)$.

Brusselator 2D. Two-dimensional Brusselator problem [39, 23]:

$$\begin{aligned} u_t &= 1 + uv^2 - 4u + \alpha\nabla^2 u, \quad x, y \in [0,1], t \in [0, 0.1], \\ v_t &= 3u - u^2 v + \alpha\nabla^2 v, \end{aligned}$$

with $\alpha = 0.2$, Dirichlet boundary conditions, and initial and boundary values

$$\begin{aligned} u &= 1 + \sin(2\pi x)\sin(2\pi y), \\ v &= 3. \end{aligned}$$

Gray-Scott 2D. Two-dimensional Gray-Scott problem [21]:

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

with $d_u = 0.2$, $d_v = 0.1$, $a = 0.04$, and $b = 0.06$, periodic boundary conditions, and initial conditions

$$\begin{aligned} u &= 1 - e^{-150(x-\frac{1}{2})^2 + (y-\frac{1}{2})^2}, \\ v &= e^{-150(x-\frac{1}{2})^2 + 2(y-\frac{1}{2})^2}. \end{aligned}$$

In all four problems the ∇^2 term was discretized using standard second-order finite differences. The resulting square uniform 2D spatial grid of size $\sqrt{N} \times \sqrt{N}$ was distributed across a 2D square grid of P processors with $\sqrt{P}$ processors per side. Each processor would thus receive a square sub-grid of the spatial domain of dimension $\sqrt{N}/\sqrt{P}$ per side. To compute the finite-difference approximation in the right-hand-side function $f(y)$ and the $J*v$ function, every function call each processor exchanges the boundary values of its spatial sub-grid with its neighboring processors using pairs of MPI_Send and MPI_Recv calls.

Both an adaptive Krylov and non-adaptive Krylov implementation of EPIRK5P1 were compared with CVODE on all four problems at four sizes per problem. All three integrators were run using variable time stepping at five absolute error tolerances ranging from $ATOL = 10^{-4}$ to $ATOL = 10^{-9}$, each tolerance differing by a factor of ten. The errors were computed using the standard 2-norm. All problems were run at four sizes, with the number of processors ranging from 16 processors to 1024 processors, each size differing by a factor of four in number of processors. At all sizes of problem, each processor was assigned a subgrid of size 80×80, the dimensions chosen to give each processor significant computational work while keeping the CPU usage of the runs within the limits of our allocation's account limit.

In general, EPIRK5P1 requires significantly fewer time steps than CVODE to achieve the same accuracy, in extreme cases requiring nearly an order of magnitude fewer steps. This meant the Jacobian matrix would be scaled far more coarsely for EPIRK5P1 than in CVODE, resulting in much higher Krylov cost per time step for the non-adaptive Krylov implementation. The adaptive Krylov implementation of EPIRK5P1 would manage the coarser scaling by subdividing the projections as described in section 5.3.2 and is less affected

by coarse scaling. However, to put the two integrators on a more even footing and facilitate more direct comparison of the rate of convergence in the Krylov iteration for the two methods, the time step sizes for EPIRK5P1 were limited to be the same size as the average step size chosen by CVODE.

All tests were performed on the Texas Advanced Computing Center (TACC) Stampede system, a 6,400 node Linux cluster connected with Mellanox FDR Infiniband. Each node contains 2 Intel Xeon E5 (Sandy Bridge) processors clocked at 2.7GHz and an Intel Xeon Phi Coprocessor. The Phi coprocessors were not used in our tests.

5.5 Numerical results

The precision diagrams for the experiments are shown in Figures 5.5 through 5.8. From the graphs we see that EPIRK5P1 is generally competitive with CVODE, particularly when implemented with adaptive Krylov. For small problem size the adaptive Krylov and non-adaptive Krylov implementations perform similarly for all four problems, but as problem size increases the non-adaptive Krylov implementation scales comparatively poorly and begins to be overtaken by CVODE. The adaptive Krylov implementation of EPIRK5P1 remains competitive. For example, in the case of the Brusselator problem in Figure 5.7, the adaptive Krylov and non-adaptive Krylov implementation perform nearly identically in the 16 processor size of the problem and significantly outperform CVODE. For the coarse error tolerance of 10^{-4}, both versions of EPIRK5P1 are over ten times as fast as CVODE. For fine tolerances the performance is closer, but even at the smallest tolerance of 10^{-9}, both cases of EPIRK5P1 require less than one fifth as much time as CVODE. By the 1024 processor size, CVODE now basically performs the same as the non-adaptive Krylov implementation of EPIRK5P1, although the adaptive Krylov implementation of EPIRK5P1 still performs the best. For an error tolerance of 10^{-4}, the adaptive version of EPIRK5P1 is over twice as fast as the other two cases. For fine tolerances the CPU times are closer, but the adaptive implementation of EPIRK5P1 still uses less than 80% the CPU time as the other two integrators.

For Krylov-based integrators, performing the Krylov iterations is the dominant CPU cost. As such the relative performance between the integrators can be understood by examining their Krylov statistics. To illustrate the cost tradeoffs between the integrators, we examine the Krylov statistics of the integrators for the Gray-Scott and ADR problem, which are the problems for which EPIRK5P1 performs comparatively worst and best respectively compared to CVODE.

We begin with the Gray-Scott problem. Table 5.3 shows the Krylov statistics for the smallest size of the Gray Scott problem. We note that the non-adaptive Krylov implementation must compute a relatively large number of Krylov vectors for the Krylov approximation of the three terms with vector hF_0, but only about a single Krylov vector for the remaining terms due to their small magnitude. In terms of computational cost, this means EPIRK5P1 must effectively perform only a single Krylov approximation each time step, as the other two approximations are of negligible cost. The adaptive Krylov implementation has the same cost structure, except the costly approximation for the hF_0 terms is split over multiple projections, lowering the total CPU cost. In contrast, each Krylov projection in CVODE is less expensive than the non-adaptive approximation for the hF_0 terms in EPIRK5P1 due to the favorable scaling of the matrix by the γ coefficient. However, it

must also compute more than one projection each time step, one for each Newton iteration. For coarse tolerances, and thus larger time steps, it must compute an average of about 1.49 projections per step. For lower error tolerances, the Newton iteration converges more quickly, within 1.29 iterations. In balance, the need to compute more nontrivial Krylov approximations per time step than EPIRK5P1 results in higher overall cost, despite the lower cost of each Krylov approximation.

Table 5.4 displays the Krylov statistics for the largest problem size. The overall statistical breakdown is very similar. EPIRK5P1 still must compute one costly Krylov approximation each time step and then two additional approximations of negligible cost. Compared to the non-adaptive Krylov approximations, CVODE must compute less expensive Krylov approximations but more than one per time step, similar to the small problem size case. The rate of convergence of the Newton iteration is about the same as in the small problem size. However, with the larger problem size the spectrum of the Jacobian is enlarged and the number of Krylov iterations needed in each approximation increases, driving the CPU time up. The larger basis sizes now result in the non-adaptive Krylov implementation of EPIRK5P1 being more expensive than CVODE in balance. For example, for the coarsest tolerance of 10^{-4}, the CPU time for the non-adaptive Krylov version of EPIRK5P1 is nearly twice that of CVODE. In the adaptive Krylov case, by splitting the approximation into lower cost projections the overall CPU cost is kept lower than CVODE for the coarser tolerances. For example, for an error tolerance of 10^{-4}, the CPU time for the adaptive Krylov EPIRK5P1 is only 40.7 seconds compared to the 59.9 seconds of CVODE. For the four finest tolerances, CVODE had lower CPU cost than even the adaptive-Krylov EPIRK5P1 for the Gray Scott problem. EPIRK5P1 took 109% the time of CVODE for a tolerance of 10^{-6} and the performance difference progressively increased up to 121% at a tolerance of 10^{-9}. For the other problems, the adaptive Krylov implementation maintained lower CPU cost than CVODE for all tolerances.

The Krylov cost structure of the integrators on the ADR problem is similar to the Gray-Scott case (and the other problems) but the costs are more modest. As before, almost all of the Krylov vectors were computed for the first hF_0 term with the remaining two terms not contributing a significant number of additional vectors. Here too CVODE needed to compute more than one projection per time step, one per Newton iteration, putting it at a disadvantage to EPIRK5P1 in terms of the number of nontrivial projections computed per time step. The basis sizes per projection were similar between the non-adaptive Krylov version of EPIRK5P1 and CVODE for this problem. Therefore due to the smaller number of projections per time step, the non-adaptive EPIRK5P1 had a net efficiency advantage over CVODE on both the small and large problem sizes.

Naturally the adaptive Krylov implementation of EPIRK5P1 fared even better, overall, and was able to outperform CVODE at all tolerances for both problem sizes. However, for the small size of problem, the basis sizes were small to the point that the Krylov adaptivity procedure did not need to split the projections, and the performance between the non-adaptive case and adaptive case was essentially the same. For the large size of the problem at coarse tolerances, the basis size for the first hF_0 term was large enough that the Krylov adaptivity algorithm split the projection into several sub-projections, giving it a performance advantage over the non-adaptive case. For example at tolerance 10^{-4}, the adaptivity procedure split the hF_0 term into an average of 3.2 projections with basis sizes only 20.2 vectors each versus the non-adaptive implementations single projection of

(a) EPIRK5P1

	Non-adaptive Krylov Absolute error tolerance						Adaptive Krylov Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
hF_0:												
Projections per step:	1	1	1	1	1	1	4.0	6.2	3.7	2.9	2.0	5.9
Vectors per projection:	23.3	21.6	18.9	16.7	14.0	11.7	10.5	7.8	8.8	8.8	9.2	2.5
Total vectors per step:	23.3	21.6	18.9	16.7	14.0	11.7	42.2	48.4	32.3	25.0	18.1	14.8
$hr(Y_1)$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.02	1.02	1.04	1.00	1.00	1.00	1.00	1.00	1.00
Total vectors per step:	1.00	1.00	1.00	1.00	1.02	1.04	1.00	1.00	1.00	1.00	1.00	1.00
$h[-2r(Y_1) + r(Y_2)]$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.0	1.01	1.02	1.00	1.00	1.00	1.00	1.00	1.00
Total vectors per step:	1.00	1.00	1.00	1.0	1.01	1.02	1.00	1.00	1.00	1.00	1.00	1.00
Time steps:	35	48	72	107	167	251	35	48	72	107	167	251
CPU time:	0.42	0.46	0.56	0.66	0.79	0.93	0.39	0.53	0.60	0.76	0.96	2.36

(b) CVODE

	Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
Newton iters per time step:	1.49	1.62	1.42	1.47	1.38	1.29
Krylov iters per Newton iter:	14.5	12.4	12.0	9.7	8.1	7.2
Time steps:	35	47	71	106	167	251
CPU time:	0.52	0.74	1.00	1.37	1.84	2.44

Table 5.3: Krylov statistics for Gray-Scott 2D with grid size 320×320.

40.0 vectors. As a result the adaptive implementation required only 5.17 seconds of time compared to the non-adaptive version's 9.57 seconds. As the error tolerances became finer, the Krylov basis sizes were reduced in size to the point that the adaptivity procedure employed fewer and fewer splits and the performance of the non-adaptive and adaptive implementations converged.

5.5.1 Impact of configuration parameters on performance

The previous experiments were conducted with the maximum time step size for the exponential integrators limited to the average step size taken by CVODE. This allowed comparison of the Krylov cost between the two types of integrators for similar scaling of the Jacobian. However, both types of integrators can accept a number of input parameters which can impact efficiency. For EPIRK5P1, an efficaciously chosen maximum time step size can keep the integrator running close to the optimal balance point between Krylov cost per time step versus the total number of time steps, as represented by the inflection point in Figure 5.1. A judiciously chosen maximum Krylov basis size can have a similar effect by forcing the time step to be lowered to maintain accuracy, thus scaling the Jacobian to a more manageable degree as well. The efficiency of CVODE is affected similarly by restrictions to time step size and Krylov basis size, but in addition the Krylov cost can be indirectly affected by restricting the number of Newton iterations, as doing so will lower the number of Krylov projections computed each time step and will force a reduction in the time step size that reduces the scaling of the Jacobian.

(a) EPIRK5P1

	Non-adaptive Krylov Absolute error tolerance						Adaptive Krylov Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
hF_0:												
Projections per step:	1	1	1	1	1	1	206	174	163	140	96.7	64.0
Vectors per projection:	198	178	160	135	110	92.5	10.0	10.2	10.2	10.1	10.1	10.2
Total vectors per step:	198	178	160	135	110	92.5	2068	1777	1662	1417	977	651
$hr(Y_1)$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.05	1.18	2.00	1.00	1.00	1.00	1.00	1.00	1.12
Total vectors per step:	1.00	1.00	1.00	1.05	1.18	2.00	1.00	1.00	1.00	1.00	1.00	1.12
$h[-2r(Y_1) + r(Y_2)]$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.00	1.03	1.16	1.00	1.00	1.00	1.00	1.00	1.02
Total vectors per step:	1.00	1.00	1.00	1.00	1.03	1.16	1.00	1.00	1.00	1.00	1.00	1.02
Time steps:	35	50	72	112	173	257	35	50	72	112	173	257
CPU time:	104	109	118	117	114	116	40.7	50.6	68.6	91.9	96.2	95.5

(b) CVODE

	Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
Newton iters per time step:	1.44	1.47	1.40	1.36	1.32	1.27
Krylov iters per Newton iter:	128	102	89.2	74.0	62.6	52.1
Time steps:	32	49	73	109	165	256
CPU time:	57.9	60.7	63.1	69.0	74.0	78.8

Table 5.4: Krylov statistics for Gray-Scott 2D with grid size 2560×2560.

An exhaustive examination of how the parameters can be set for optimal performance is outside the scope of this paper but we give a single example of how limiting the maximum time step can reduce the CPU time for both the non-adaptive Krylov and adaptive Krylov implementations of EPIRK5P1. When set to an absolute error tolerance of 10^{-4} on the 2560×2560 size of the Gray-Scott problem, computed with a maximum time step size equal to the average step size taken for CVODE as $h_{max} = 0.0029$, the non-adaptive Krylov implementation of EPIRK5P1 required about 104 seconds of time to compute 35 time steps, and the adaptive Krylov implementation needed 40.7 seconds for the same number of steps, as shown in Table 5.4. When the maximum time step size was further reduced to $h_{max} = 0.0006$, the non-adaptive Krylov implementation of EPIRK5P1 used only 27.4 seconds of time to compute 167 time steps while the adaptive Krylov implementation required only about 23.0 seconds for the same number of steps. This means the non-adaptive Krylov implementation took only 26% of the total CPU time in the reduced step size case compared to the prior case, and the adaptive Krylov implementation just 57%, giving significant reductions in both cases.

While it is not surprising that choosing a step size close to the optimum balance between Krylov cost per step versus total computed time steps gives better efficiency for the non-adaptive Krylov implementation, it is interesting to note that the same holds true for the adaptive Krylov case as well. In principle, an adaptive Krylov procedure which makes optimal splits should give the best possible performance without the need for manually modulating the time step size. The fact that the CPU time could be reduced by nearly half in the Gray-Scott example by lowering the maximum time step size is an indication of

(a) EPIRK5P1

	Non-adaptive Krylov Absolute error tolerance						Adaptive Krylov Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
hF_0:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	6.2	4.6	4.1	4.0	4.2	4.6	5.7	4.1	3.4	3.6	3.8	4.1
Total vectors per step:	6.2	4.6	4.1	4.0	4.2	4.6	5.7	4.1	3.4	3.6	3.8	4.1
$hr(Y_1)$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.24	1.57	1.81	1.00	1.00	1.00	1.00	1.26	1.78
Total vectors per step:	1.00	1.00	1.00	1.24	1.57	1.81	1.00	1.00	1.00	1.00	1.26	1.78
$h[-2r(Y_1) + r(Y_2)]$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.00	1.41	1.80	1.00	1.00	1.00	1.00	1.00	1.48
Total vectors per step:	1.00	1.00	1.00	1.00	1.41	1.80	1.00	1.00	1.00	1.00	1.00	1.48
Time steps:	125	200	266	409	527	658	125	200	266	409	527	658
CPU time:	0.17	0.20	0.26	0.39	0.56	0.76	0.19	0.26	0.33	0.51	0.69	0.95

(b) CVODE

	Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
Newton iters per time step:	1.82	1.73	1.59	1.60	1.82	1.62
Krylov iters per Newton iter:	6.8	5.3	4.7	3.6	2.9	2.8
Time steps:	124	200	265	409	525	660
CPU time:	1.11	1.61	2.25	3.07	3.91	4.56

Table 5.5: Krylov statistics for ADR 2D with grid size 320×320.

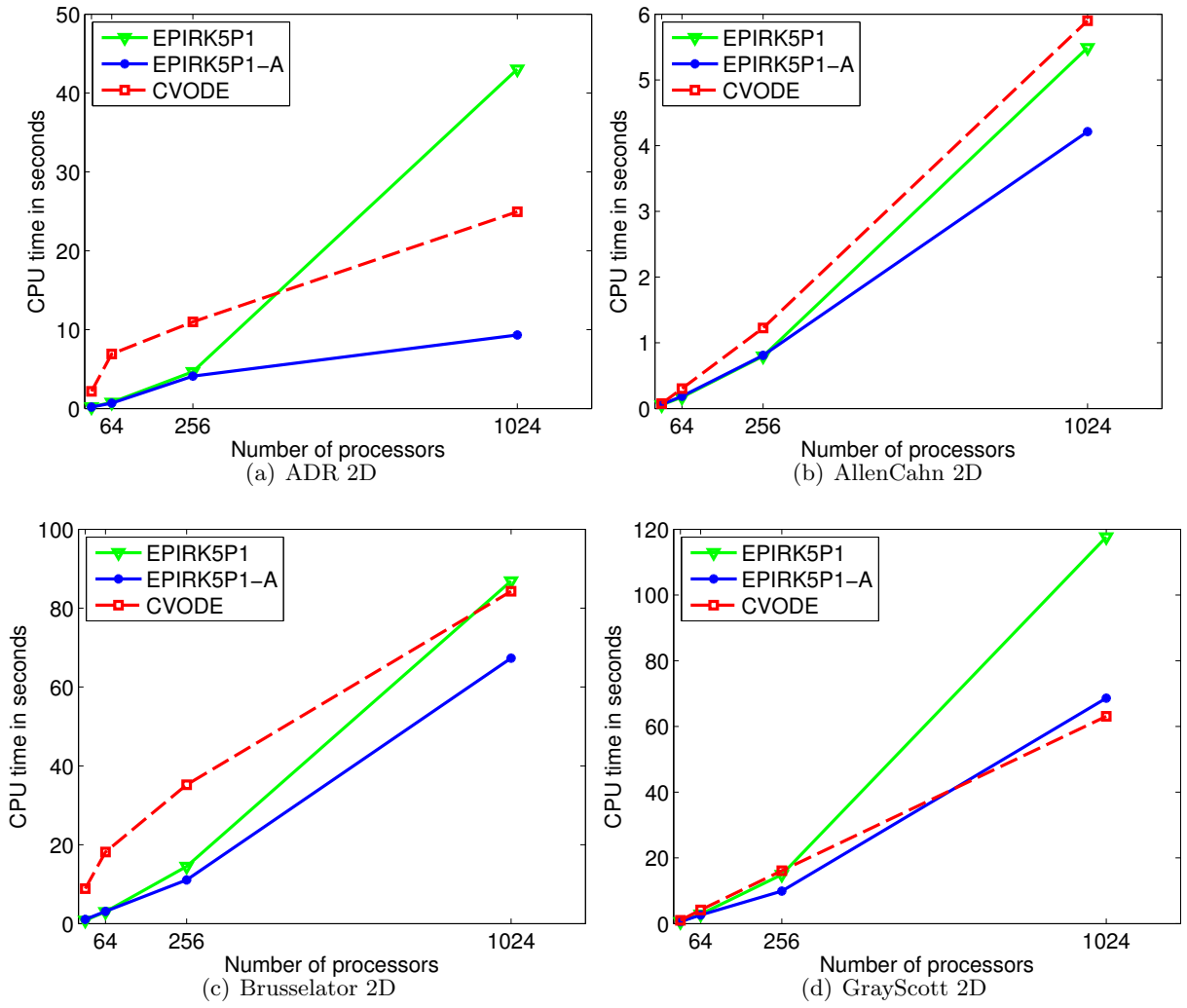
room for improvement in the algorithm.

5.5.2 Impact of Krylov adaptivity on scalability

Because the number of Krylov iterations increases with problem size due to the widening spectrum of the Jacobian, the CPU cost increases with problem size even if the grid density per processor remains constant. As such, Krylov-based integrators generally scale poorly with problem size without preconditioning, although Krylov adaptivity can improve scalability significantly. To illustrate this, Figure 5.4 shows the relationship of problem size to CPU time for each of the four problems. For all four problems, the CPU times were for the case where the absolute tolerance is 10^{-6}, but the curves are similar in the other cases. We see that for all problems, the performance of the non-adaptive Krylov implementation of EPIRK5P1 generally scales poorly with problem size, having a much steeper slope compared with the adaptive Krylov implementation of EPIRK5P1 or CVODE. In contrast, the adaptive Krylov version of EPIRK5P1 scales considerably better, generally scaling the best of all three integrators.

5.6 Conclusions and future work

This paper describes the implementation for what is to our knowledge the first parallel implementation of a suite of Krylov-based exponential integrators and gives some initial performance results for an adaptive Krylov-based implementation of the EPIRK5P1 solver.



(a) ADR 2D (b) AllenCahn 2D

(c) Brusselator 2D (d) GrayScott 2D

Figure 5.4: Algorithmic scaling of the integrators with problem size. Solutions computed to an absolute tolerance of 10^{-6} in all cases.

(a) EPIRK5P1

	Non-adaptive Krylov Absolute error tolerance						Adaptive Krylov Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
hF_0:												
Projections per step:	1	1	1	1	1	1	3.20	1.88	1.84	1.40	1.02	1.00
Vectors per projection:	49.0	30.1	29.0	21.8	17.0	13.2	20.2	19.8	19.6	19.2	17.3	13.6
Total vectors per step:	49.0	30.1	29.0	21.8	17.0	13.2	64.7	37.1	36.1	26.8	17.7	13.6
$hr(Y_1)$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.14	1.35	1.46	1.00	1.00	1.00	1.00	1.00	1.12
Total vectors per step:	1.00	1.00	1.00	1.14	1.35	1.46	1.00	1.00	1.00	1.00	1.00	1.12
$h[-2r(Y_1) + r(Y_2)]$:												
Projections per step:	1	1	1	1	1	1	1.0	1.0	1.0	1.0	1.0	1.0
Vectors per projection:	1.00	1.00	1.00	1.00	1.16	1.34	1.00	1.00	1.00	1.00	1.00	1.00
Total vectors per step:	1.00	1.00	1.00	1.00	1.16	1.34	1.00	1.00	1.00	1.00	1.00	1.00
Time steps:	124	235	274	435	658	1022	124	235	274	435	658	1022
CPU time:	9.57	6.53	7.10	6.48	6.94	7.52	5.17	5.65	6.25	7.45	7.47	7.79

(b) CVODE

	Absolute error tolerance					
	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
Newton iters per time step:	1.82	1.52	1.50	1.52	1.61	1.59
Krylov iters per Newton iter:	43.1	27.5	26.5	16.8	11.6	8.3
Time steps:	123	234	274	435	656	1021
CPU time:	34.1	28.7	31.2	31.5	33.7	34.4

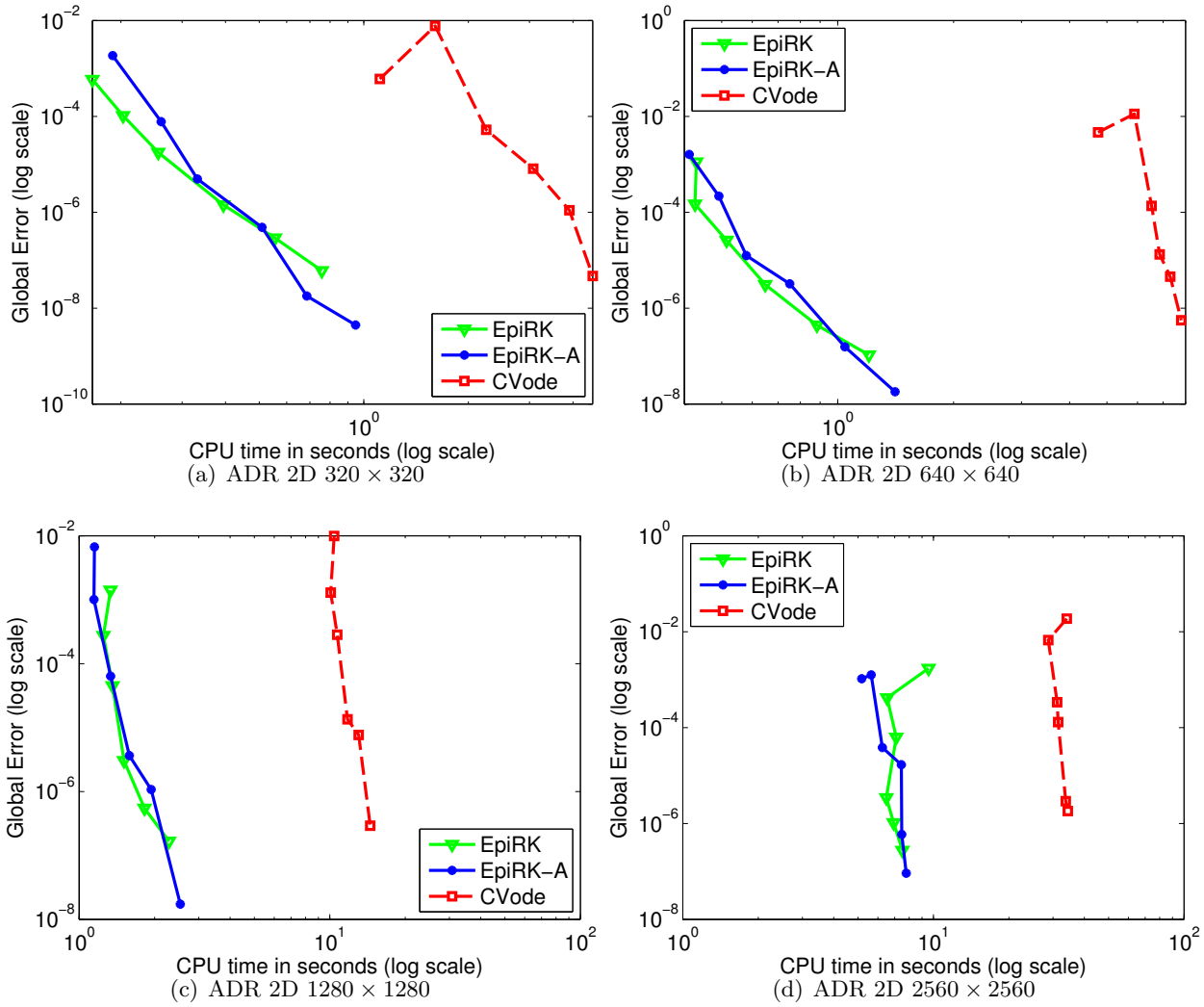
Table 5.6: Krylov statistics for ADR 2D with grid size 2560×2560.

Our experiments show EPIRK5P1 to be on par, both in terms of computational efficiency and algorithmic scalability, with a production implementation of an implicit Newton-Krylov solver on a set of stiff benchmark problems when integrated without preconditioning. We discussed some of the features of the software suite, which include its extensibility to new exponential schemes and alternate ways of approximating the $\varphi_k(hJ)v$ terms, and its ability to accept problems written for CVODE. Utilizing these features, in the future we intend to extend the suite to include newly developed exponential methods, and to test the integrators on a large scale problem of current scientific interest.

While the performance results are encouraging, further questions must be addressed. The comparisons were made on 2D test problems, and it remains to be seen how the exponential integrator performs on more complex application problems. Furthermore the problems were tested without preconditioning. While our results show Krylov adaptivity can significantly improve the algorithmic scalability of the method, development of preconditioning for exponential integrators will be important for problems for which efficient preconditioners already exist for Newton-Krylov implicit solvers. Some recent work on split [56] and hybrid exponential integrators might offer preconditioning-like solutions for exponential methods and we plan to investigate these methods in the future.

5.7 Acknowledgements

This work was supported by a grant from the National Science Foundation, Computational Mathematics Program, under Grant No. 1115978.



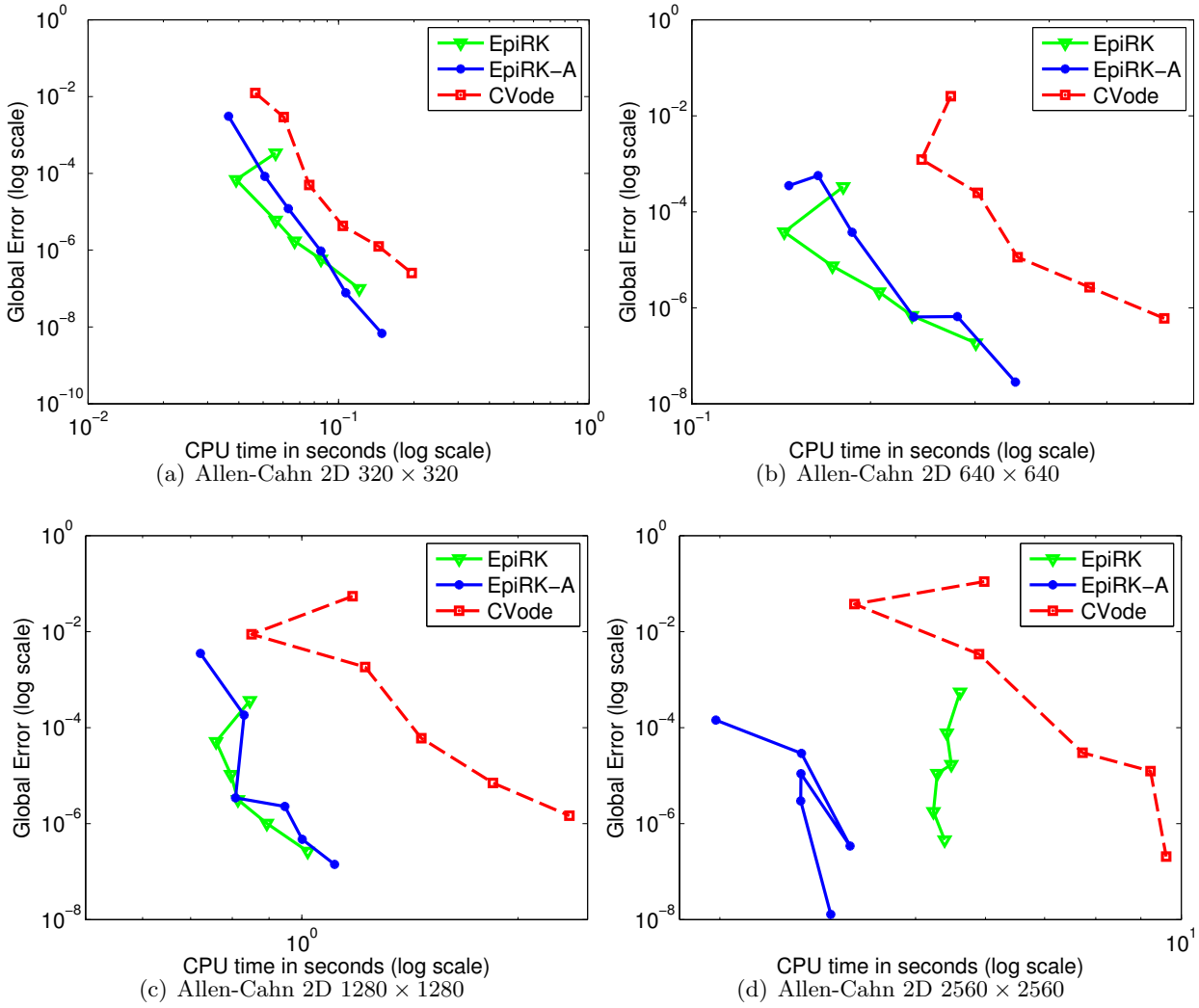
(a) ADR 2D 320×320

(b) ADR 2D 640×640

(c) ADR 2D 1280×1280

(d) ADR 2D 2560×2560

Figure 5.5: Precision diagrams comparing nonadaptive and adaptive Krylov implementations of EPIRK5P1 against CVODE for ADR2d.



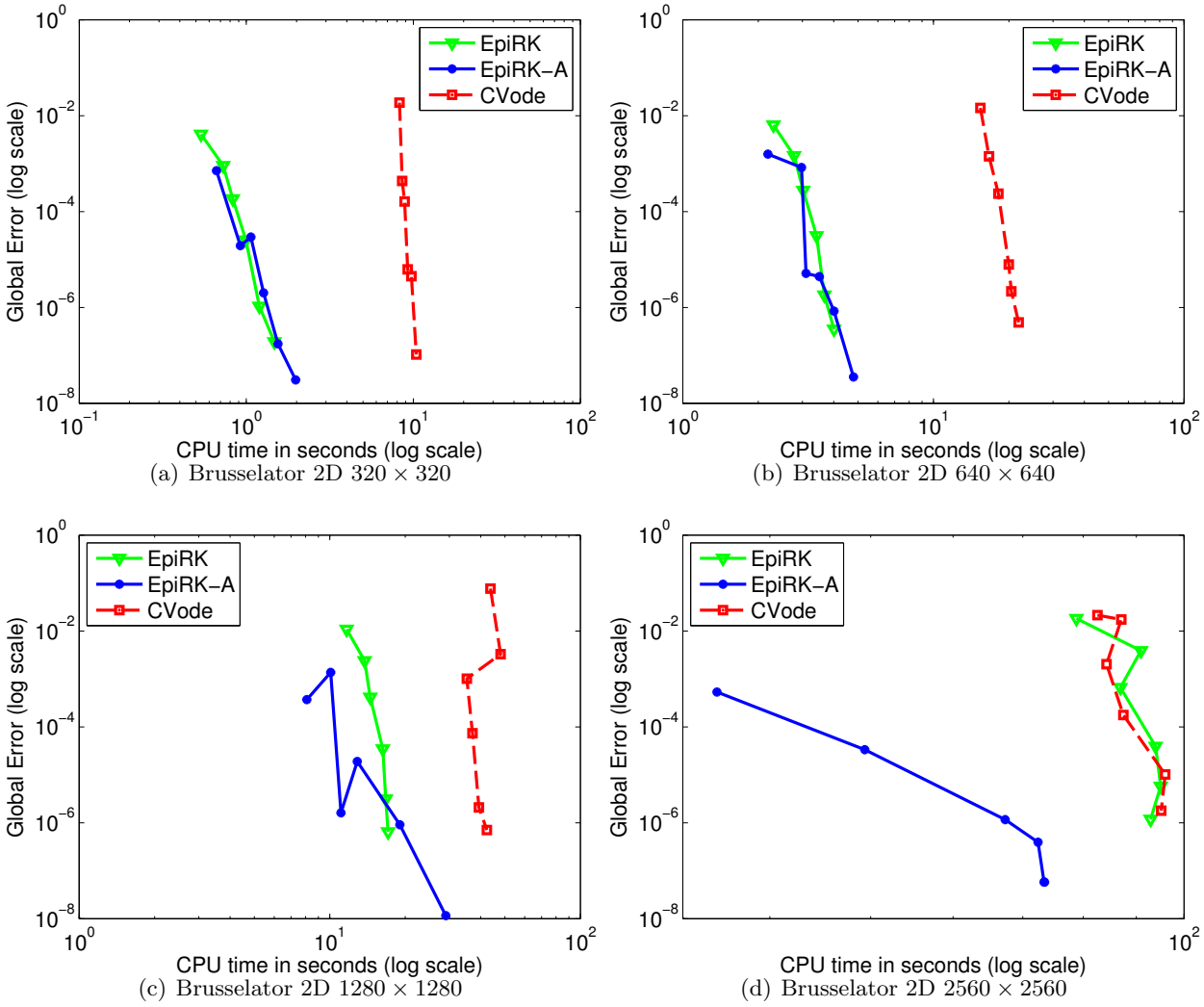
(a) Allen-Cahn 2D 320×320

(b) Allen-Cahn 2D 640×640

(c) Allen-Cahn 2D 1280×1280

(d) Allen-Cahn 2D 2560×2560

Figure 5.6: Precision diagrams comparing nonadaptive and adaptive Krylov implementations of EPIRK5P1 against CVODE for AllenCahn2d.



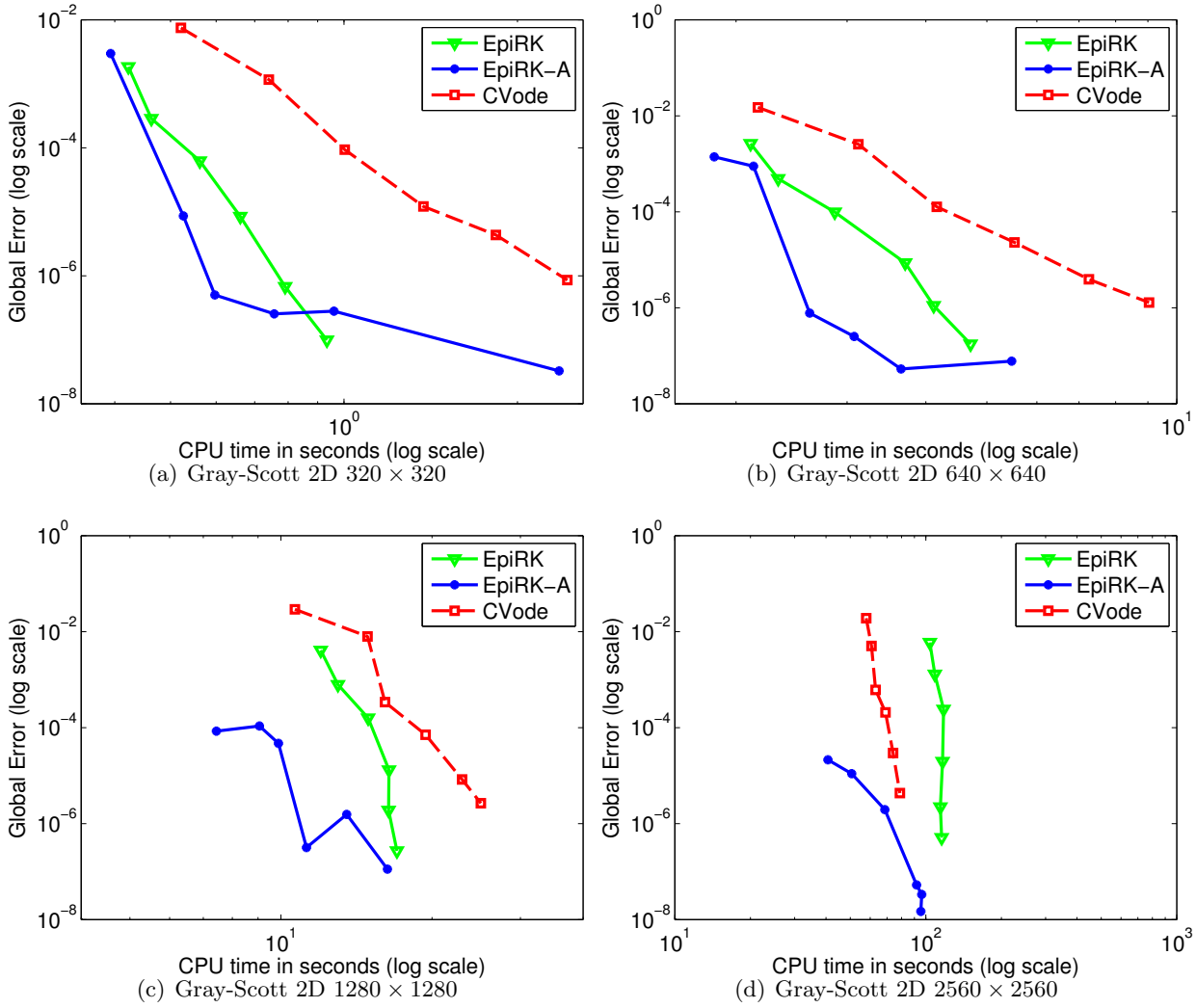
(a) Brusselator 2D 320×320

(b) Brusselator 2D 640×640

(c) Brusselator 2D 1280×1280

(d) Brusselator 2D 2560×2560

Figure 5.7: Precision diagrams comparing nonadaptive and adaptive Krylov implementations of EPIRK5P1 against CVODE for Brusselator2d.



(a) Gray-Scott 2D 320×320

(b) Gray-Scott 2D 640×640

(c) Gray-Scott 2D 1280×1280

(d) Gray-Scott 2D 2560×2560

Figure 5.8: Precision diagrams comparing nonadaptive and adaptive Krylov implementations of EPIRK5P1 against CVODE for GrayScott2d.

6 Conclusions

6.0.1 Summary

In this thesis we conducted a performance study of Krylov-based exponential integrators with the goal of determining how to improve their computational efficiency. Various design criteria of exponential schemes were investigated through thorough performance benchmarking and some of the lessons were applied to the construction of new exponential methods of higher efficiency. The construction of the new schemes was done in the context of the new EPIRK class of exponential integrators, which provides a high degree of flexibility for tailoring the methods for performance. New methods of note are the EPIRK5S3 method which has fifth order accuracy for lower computational cost than current fourth order methods, and the EPIRK5P1 and EPIRK5P2 integrators which provide similar performance benefits while also allowing the methods to employ a Krylov adaptivity algorithm which further improves efficiency quite substantially.

Detailed performance comparisons with Krylov-based implicit methods were also presented. The results showed that the exponential-like $\varphi_k(A)$ matrix functions which constitute exponential schemes give an inherent performance advantage over Krylov-based implicit methods, which must compute a more costly matrix rational function. Various structural tradeoffs between exponential and implicit methods were discussed, and it was demonstrated that overall exponential methods can perform favorably compared with implicit methods.

A software suite of parallel exponential integrators was introduced and described. The software is designed to allow easy extensibility to new exponential schemes and alternate techniques for the approximation of the $\varphi_k(A)$ functions. Performance of the Krylov-adaptive EPIRK5P1 method from the suite was compared with the popular Newton-Krylov CVODE solver and shown to perform favorably on unpreconditioned problems across a range of problem sizes. Krylov adaptivity was shown to be particularly useful in managing the growth of Krylov cost with increasing problem size. To our knowledge, our software provides the first parallel implementation of Krylov-based exponential integrators. Performance comparisons with CVODE represent the first preliminary performance study of exponential integrators compared to Krylov-based implicit solvers on parallel-scale problems.

6.0.2 Future work

There is still much to be done if exponential integrators are to become commonly used for large-scale scientific problems. The performance testing in this thesis was done using a number of stiff benchmark problems, but larger and more sophisticated problems need to be tested if exponential integrators are to gain the attention of practitioners. While we discussed some first results on the scalability of Krylov-based exponential integrators, a more extensive examination of how exponential integrators scale to very large problem sizes is particularly important for parallel scale computing. Further performance optimization of

exponential schemes and their implementations should continue. In light of these needs we briefly note some ways what has been discussed in this thesis can be extended and improved.

We discussed the idea of tailoring the coefficients of a scheme to increase the efficiency of a method. Two examples were the EPIRK5S3 and EPIRK5P1 schemes. While the coefficients improved the performance of the methods over previously derived schemes, the constants are likely not optimal. Derivation of even better schemes might offer significant performance improvement.

We showed that the use of Krylov adaptivity improves the efficiency of exponential integrators considerably when large time step sizes are taken, but the algorithm could be improved. The discussion in section 5.5.1 showed that even when using Krylov adaptivity, imposing a maximum time step size below that needed to achieve the error tolerance could still further improve the efficiency, nearly halving the CPU time over adaptivity alone in some cases. An ideal adaptivity algorithm would partition the $\varphi_k(hJ)v$ terms optimally regardless of the time step size, and the improved CPU times seen in section 5.5.1 would be achieved by a more ideal algorithm without manual intervention. However, there are significant challenges to improving the adaptivity algorithm in its current form. We saw from our performance results shown in Table 5.4, that for large problems the adaptivity procedure may need to split the $\varphi_k(hJ)v$ terms across hundreds of sub-projections. The current adaptivity algorithm chooses the scaling of the matrix in each sub-projection without consideration of the other sub-projections, effectively assuming the matrix scaling chosen in the current sub-projection will be equally good for all other sub-terms. Empirical tests show this assumption is false. However determining a partitioning of the $\varphi_k(hJ)v$ terms which is globally optimal, with the matrix possibly scaled differently in each sub-term, would be mathematically difficult. Furthermore, the error control is done to a specified tolerance for each sub-projection, but there is currently no accounting of how errors propagate from sub-term to sub-term. Controlling the error of the final result across hundreds of sub-terms would also be difficult. Rather than try to improve the algorithm by partitioning the possibly hundreds of sub-projections more optimally, a more practical solution might be to modulate the time step size directly. The Krylov adaptivity algorithm can be parameterized to allow no more than K sub-projections for each term. If the adaptivity procedure finds it needs to take more than the allowed number of sub-projections, the step size could be reduced until that is no longer the case. In cases where the number of splits needed falls below K, the step size would be relaxed up to the size needed to achieve the error tolerance for the time step. This approach is nothing more than an automation of the parameter configuration experiment discussed in section 5.5.1, and the expected improvement in efficiency would be the same as found in those results. Experience with those tests shows that heuristically limiting K to just two or three would give the best results.

Scalability is a primary concern for large scale computing, and is a particular challenge for Krylov-based integrators. We discussed in section 5.5.2 how increasing problem size causes the spectrum of the Jacobian to widen, resulting in the number of Krylov vectors needed per term to increase. Our tests showed that Krylov adaptivity can provide a significant improvement to the scalability of Krylov-based exponential integrators. The benefit of adaptivity stems from the observation that the cost per Krylov vector increases quadratically with the basis size of each projection, but by subdividing the $\varphi_k(hJ)v$ terms into multiple sub-projections with smaller basis sizes, adaptivity prevents the cost per vector from growing too large. However adaptivity cannot lower the total number of Krylov vectors that must

be computed, it can only prevent the cost per vector from growing by spreading the vectors over multiple smaller sized projections. A methodology for mitigating the increase in the number of Krylov vectors needed per projection as the spectrum grows with problem size would be desirable. For Krylov-based implicit integrators, preconditioning is the primary means for minimizing the number of Krylov vectors which must be computed, and is the chief means of making such methods scalable. While there have been some early results [47, 71], currently no efficient preconditioning strategies exist for exponential integrators. Development of effective preconditioning could significantly improve the scalability of exponential integrators and make them competitive with implicit integrators on problems for which good preconditioning approaches already exist.

Implicit-explicit (IMEX) methods can be efficient for problems which can be split into stiff and non-stiff portions. Semilinear problems of the form $y' = Ly + N(y)$ where the stiffness is primarily constrained to the L operator, such as many diffusion-reaction problems, are common examples of such systems. Since the problem is stiff, integrating the problem with an explicit integrator is typically cost prohibitive, but by treating the stiff portion with implicit time integration and the non-stiff portion with explicit integration IMEX methods can save cost over applying a fully implicit method to the problem. As a tradeoff, IMEX methods typically have reduced stability compared to fully implicit integration. Currently there is work to develop hybrid implicit-exponential integrators [70], which replace the explicit treatment of the non-stiff portion with exponential integration. While exponential methods are generally more computationally costly than classical explicit integration, such hybrid methods may prove efficient for problems where the limited stability of IMEX methods requires smaller time step sizes than needed for accuracy. An advantage of such methods would be that preconditioning strategies for the implicit portion of IMEX methods would carry over to implicit-exponential methods, possibly providing a middle ground to finding preconditioning approaches for exponential methods directly.

There are a variety of other means for approximating the $\varphi_k(hJ)v$ terms besides Krylov techniques. Some examples are polynomial approximation, such as Chebyshev [52] or Leja point approximation [10], improvements to Taylor approximation [2], and contour integral approximation [33, 59, 73]. Depending on the problem, some of these techniques may prove to be more efficient than Krylov approximation, and in some cases and may scale better on large parallel machines. Currently performance comparisons between approaches are limited [7, 8] however, particularly scalability studies for parallel implementations. Our software is designed to accommodate alternative methods for approximating the $\varphi_k(hJ)v$ terms and we hope to use it for comparison of approaches.

Finally there needs to be more studies of the application of exponential integrators to large-scale scientific problems of current interest. The majority of performance analysis so far has been done using benchmarking problems. There have been a number of cases of applying exponential integrators to scientific applications on the single-processor scale, ranging from magnetohydrodynamics [65, 40] to option pricing [64, 38], but to our knowledge there have been none on the parallel scale. Better understanding of how exponential integrators perform on very large problems of current interest in high performance computing would help popularize them in the scientific community.

Bibliography

[1] A. H. Al-Mohy and N. J. Higham, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31(3) (2009), pp. 970–989.

[2] ——, *Computing the action of the matrix exponential, with an application to exponential integrators*, SIAM J. Sci. Comp., 33(2) (2011), pp. 488–511.

[3] W. E. Arnoldi, *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., 9 (1951), pp. 17–29.

[4] P. Bates, S. Brown, and J. Han, *Numerical analysis for a nonlocal Allen-Cahn equation*, Int. J. Numer. Anal. Mod., 6(1) (2009), pp. 33–49.

[5] G. Beylkin, J. M. Keiser, and L. Vozovoi, *A new class of time discretization schemes for the solution of nonlinear PDEs*, J. Comput. Phys., 147 (1998), pp. 362–387.

[6] J. C. Butcher, *Trees, B-series and exponential integrators*, IMA J.Numer. Anal., 30 (2009), pp. 131–140.

[7] M. Caliari, L. Bergamaschi, A. Martinez, and M. Vianello, *Comparing Leja and Krylov approximations of large scale matrix exponentials*, in of LNCS, Springer, 2006, pp. 685–692.

[8] M. Caliari, P. Kandol, A. Ostermann, and S. Rainer, *Comparison of various methods for computing the action of the matrix exponential*, (submitted).

[9] M. Caliari and A. Ostermann, *Implementation of exponential Rosenbrock-type integrators*, Appl. Numer. Math., 59 (2009), pp. 568–581.

[10] M. Caliari, M. Vianello, and L. Bergamaschi, *Interpolating discrete advection-diffusion propagators at Leja sequences*, J. Comput. Appl. Math., 172 (2004), pp. 79–99.

[11] M. Calvo and C. Palencia, *A class of explicit multistep exponential integrators for semilinear problems*, Numer. Math., 102 (2006), pp. 367–381.

[12] J. Certaine, *The solution of ordinary differential equations with large time constants*, Wiley, 1967.

[13] S. Cox and P. Matthews, *Exponential time differencing for stiff systems*, J. Comput. Phys., 176 (2002), pp. 430–455.

[14] P. E. Crouch and R. Grossman, *Numerical integration of ordinary differential equations on manifolds*, J. Nonlinear Sci., 3 (1993), pp. 1–33.

[15] H. A. V. der Vorst, *An iterative solution method for solving* $\mathbf{f(a)x} = \mathbf{b}$ *using Krylov subspace information obtained for the symmetric positive definite matrix* $\mathbf{a}$, J. Comput. Appl. Math., 18 (1987), pp. 249–263.

[16] G. Dimarco and L. Pareschi, *Exponential Runge-Kutta methods for stiff kinetic equations*, SIAM J. Numer. Anal., 49 (2011), pp. 2057–2077.

[17] B. O. E. Celledoni, A. Marthinsen, *Commutator-free Lie group methods*, FGCS, 19 (2003), pp. 341–352.

[18] A. Friedli, *Verallgemeinerte Runge-Kutta Verfahren zur Lósung steifer Differentialgleichungssysteme*, Springer, 1978.

[19] R. A. Friesner, L. S. Tuckerman, B. C. Dornblaser, and T. V. Russo, *A method for exponential propagation of large systems of stiff nonlinear differential equations*, J. Sci. Comput., 4 (1989), pp. 327–354.

[20] E. Gallopoulos and Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods*, SIAM J. Sci. Stat. Comp, 13 (1992), pp. 1236–1264.

[21] P. Gray and S. K. Scott, *Autocatalytic reactions in the isothermal continuous stirred tank reactor.*, Chem. Engng. Sci., 39 (1984), pp. 1087–1097.

[22] E. Hairer, S. P. Norsett, and G. Wanner, *Solving ordinary differential equations I: Nonstiff problems*, Springer, 2nd ed., 2004.

[23] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems*, Springer, 2nd ed., 2004.

[24] N. Higham, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 1179–1193.

[25] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. Woodward, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*, ACM TOMS, 31(3) (2005), pp. 363–396.

[26] M. Hochbruck and C. Lubich, *On Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 34 (1997), pp. 1911–1925.

[27] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput., 19 (1998), pp. 1552–1574.

[28] M. Hochbruck and A. Ostermann, *Explicit exponential Runge-Kutta methods for semilinear parabolic problems*, SIAM J. Numer. Anal., 43 (2005), pp. 1069–1090.

[29] ——, *Exponential Runge-Kutta methods for parabolic problems*, Appl. Numer. Math., 53 (2005), pp. 323–339.

[30] ——, *Exponential integrators of Rosenbrock-type*, Oberwolfach Reports, 3 (2006), pp. 1107–1110.

[31] ——, *Exponential integrators*, Acta Numer., 19 (2010), pp. 209–286.

[32] M. Hochbruck, A. Ostermann, and J. Schweitzer, *Exponential Rosenbrock-type methods*, SIAM J. Numer. Anal., 47 (2009), pp. 786–803.

[33] A. K. Kassam and L. N. Trefethen, *Fourth-order time stepping for stiff PDEs.*, SIAM J. Sci. Comput., 26 (2005), pp. 1214–1233.

[34] D. A. Knoll and D. E. Keyes, *Jacobian-free Newton-Krylov methods: a survey of approaches and applications*, J. Comp. Phys., 193 (2004), pp. 357–397.

[35] S. Krogstad, *Generalized integrating factor methods for stiff PDEs*, J. Comp. Phys., 203 (2005), pp. 72–88.

[36] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, Wiley, 2nd ed., 1991.

[37] J. D. Lawson, *Generalized Runge-Kutta processes for stable systems with large Lipschitz constants*, SIAM J. Numer. Anal., 4 (1967), pp. 372–380.

[38] S. T. Lee, X. Liu, and H. Sun, *Fast exponential time integration scheme for option pricing with jumps*, Numer. Linear Algebra Appl., 19 (2010), pp. 87–101.

[39] R. Lefever and G. Nicolis, *Chemical instabilities and sustained oscillations.*, J. Theor. Biol., 3 (1971), pp. 267–284.

[40] P. W. Livermore, *An implementation of the exponential time differencing scheme to the magnetohydrodynamic equations in a spherical shell*, J. Comp. Phys., 220 (2007), pp. 824–838.

[41] J. Loffeld and M. Tokman, *Comparative performance of exponential, implicit, and explicit integrators for stiff systems of ODEs*, J. Comput. Appl. Math., 241 (2013), pp. 45–67.

[42] A. Martinez, L. Bergamaschi, M. Caliari, and M. Vianello, *A massively parallel exponential integrator for advection-diffusion models*, J. Comput. Appl. Math., 231 (2009), pp. 82–91.

[43] S. Maset and M. Zennaro, *Unconditional stability of explicit exponential Runge-Kutta methods for semi-linear ordinary differential equations*, Math. Comput., 78 (2009), pp. 957–967.

[44] B. V. Minchev, *Lie group integrators for non-autonamous frozen vector fields*, Int. J. Comput. Sci. Eng., 3 (2007), pp. 287–294.

[45] B. V. Minchev and W. M. Wright, *A review of exponential integrators for first order semi-linear problems*, Technical Report, (2005), pp. 1–44.

[46] C. B. Moler and C. F. V. Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.

[47] I. Moret and P. Novati, *RD-rational approximations of the matrix exponential*, BIT, 44 (2004), p. 595615.

[48] H. Munthe-Kaas, *High order Runge-Kutta methods on manifolds*, APPL. NUMER. MATH, 29 (1999), pp. 115–127.

[49] A. Nauts and R. E. Wyatt, *New approach to many-state quantum dynamics: the recursive-residue-generation method*, Phys. Rev. Lett., 51 (1983), pp. 2238–2241.

[50] J. Niesen and W. M. Wright, *Algorithm 919: A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators*, ACM Trans. Math. Softw., 38 (2012), pp. 22:1–22:19.

[51] P. Novati, *On the construction of restricted-denominator exponential W-methods*, J. Comput. Appl. Math., 212 (2008), pp. 86–101.

[52] ——, *Polynomial methods for the computation of functions of large unsymmetric matrices*, PhD thesis, University of Padova, 2011.

[53] A. Ostermann, M. Thalhammer, and W. M. Wright, *A class of explicit exponential general linear methods*, BIT Num. Math., 46 (2006), pp. 409–431.

[54] T. J. Park and J. C. Light, *Unitary quantum time evolution by iterative Lanczos reduction*, J. Chem. Phys., 85 (1986), pp. 5870–5876.

[55] D. A. Pope, *An exponential method of numerical integration of ordinary differential equations*, Comm. ACM, 6 (1963), pp. 491–493.

[56] G. Rainwater and M. Tokman, *A new class of split exponential propagation iterative methods of Runge-Kutta type (sEPIRK) for semilinear systems of ODEs*, (submitted).

[57] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 29 (1992), pp. 209–228.

[58] ——, *Iterative methods for sparse linear systems*, PWS Publishing Company, 1996.

[59] T. Schmelzer and L. N. Trefethen, *Evaluating matrix functions for exponential integrators via Carathéodory-Fejér approximation and contour integrals*, ETNA, 29 (2007), pp. 1–18.

[60] B. A. Schmitt and R. Weiner, *Matrix-free W-methods using a multiple Arnoldi iteration.*, Appl. Numer. Math., 18 (1995), pp. 307–320.

[61] J. A. Sherratt, *On the form of smooth-front travelling waves in a reaction-diffusion equation with degenerate nonlinear diffusion*, Math Model. Nat. Phenom., 5(5) (2010), pp. 63–78.

[62] R. Sidje, *Expokit: A software package for computing matrix exponentials*, ACM Trans. Math. Softw., 24 (1998), pp. 130–156.

[63] B. Skaflestad and W. M. Wright, *The scaling and modified squaring method for matrix functions related to the exponential*, Appl. Numer. Math., 59 (2009), pp. 783–799.

[64] D. Y. Tangman, A. Gopaul, and M. Bhuruth, *Exponential time integration and Chebychev discretization schemes for fast pricing of options*, Appl. Numer. Math., 58 (2008), pp. 1309–1319.

[65] M. Tokman, *Magnetohydrodynamics modeling of solar magnetic arcades using exponential propagation methods*, PhD thesis, California Institute of Technology, 2001.

[66] ——, *Efficient integration of large stiff systems of ODEs with exponential propagation iterative (EPI) methods*, J. Comp. Phys., 213 (2006), pp. 748–776.

[67] ——, *A new class of exponential propagation iterative methods of Runge-Kutta type (EPIRK)*, J. Comp. Phys., 230 (2011), pp. 8762–8778.

[68] M. Tokman and J. Loffeld, *Efficient design of exponential-Krylov integrators for large scale computing*, Procedia Computer Science, 1 (2010), pp. 229–237.

[69] M. Tokman, J. Loffeld, and P. Tranquilli, *New adaptive exponential propagation iterative methods of Runge-Kutta type (EPIRK)*, SIAM J. Sci. Comput., 34(5) (2012), p. A2650A2669.

[70] M. Tokman and G. Rainwater, *New hybrid implicit-exponential integrators for large stiff systems of ODEs*, (submitted).

[71] J. v. d. Eshof and M. Hochbruck, *Preconditioning Lanczos approximations to the matrix exponential*, SIAM J. Sci. Comp., 27 (2005), p. 14381457.

[72] N. Vaissmoradi, A. Malek, and S. Momeni-Masuleh, *Error analysis and applications of the Fourier-Galerkin Runge-Kutta schemes for high-order stiff PDEs*, J. Comput. Appl. Math., 231 (2009), pp. 124–133.

[73] J. A. C. Weideman, *Improved contour integral methods for parabolic PDEs*, IMA J. Numer. Anal., 30 (2010), pp. 334–350.

[74] R. Weiner and B. A. Schmitt, *Order results for Krylov-W-methods*, Computing, 61 (1998), p. 6989.

[75] R. Weiner, B. A. Schmitt, and H. Podhaisky, *ROWMAP - a ROW-code with Krylov techniques for large stiff ODEs*, Appl. Numer. Math., 25 (1997), pp. 303–319.