

Irksome: Automating Runge–Kutta time-stepping for finite element methods

PATRICK E. FARRELL, University of Oxford, UK

ROBERT C. KIRBY, Baylor University, USA

JORGE MARCHENA-MENÉNDEZ, Baylor University, USA

While implicit Runge–Kutta methods possess high order accuracy and important stability properties, implementation difficulties and the high expense of solving the coupled algebraic system at each time step are frequently cited as impediments. We present Irksome, a high-level library for manipulating UFL (Unified Form Language) expressions of semidiscrete variational forms to obtain UFL expressions for the coupled Runge–Kutta stage equations at each time step. Irksome works with the Firedrake package to enable the efficient solution of the resulting coupled algebraic systems. Numerical examples confirm the efficacy of the software and our solver techniques for various problems.

CCS Concepts: • **Mathematics of computing** → **Mathematical software**; *Partial differential equations*; • **Computing methodologies** → *Hybrid symbolic-numeric methods*; • **Software and its engineering** → *Source code generation*;

ACM Reference Format:

Patrick E. Farrell, Robert C. Kirby, and Jorge Marchena-Menéndez. 2021. Irksome: Automating Runge–Kutta time-stepping for finite element methods. *ACM Trans. Math. Softw.* 1, 1 (February 2021), 26 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Many successful high-level finite element packages provide a domain-specific language that allows users to succinctly specify a partial differential equation (PDE) in mathematical syntax and obtain an efficient implementation. UFL, the Unified Form Language [Alnæs et al. 2014], provides such a Python-based domain-specific language. It was originally introduced as part of FEniCS [Logg et al. 2012] but has since been adopted by both Firedrake [Rathgeber et al. 2016] and DUNE-FEM [Dedner et al. 2010].

In addition to enabling the generation of low-level finite element code, such an abstract interface also provides opportunities for “outer-loop” development of algorithms built on top of PDE solves. For example, dolfin-adjoint [Farrell et al. 2013] allows users to specify forward models and cost functionals in UFL and then automatically derives adjoint and tangent-linear models, key ingredients in data assimilation, optimization, and sensitivity analysis. The more recent hIPPYlib [Villa et al. 2018] provides many of these features but also includes a framework for Bayesian optimization and certain efficient Hessian approximations. RBniCS [Ballarin et al. 2015; Hesthaven et al. 2016] provides several approaches to reduced order modeling, built on top of FEniCS’ interface.

Despite its richness of support for diverse spatial discretizations of various kinds and orders, UFL lacks a comparable abstraction for time-stepping. Although early versions of DOLFIN interfaced to multi-adaptive temporal Galerkin methods [Logg 2003], users wishing to solve time-dependent problems typically write their own time-stepping loops with relatively elementary methods. A FEniCS-based library for time abstractions was developed in [Maddison and

Authors’ addresses: Patrick E. Farrell, University of Oxford, Mathematical Institute, Woodstock Road, OX2 6GG, UK, patrick.farrell@maths.ox.ac.uk; Robert C. Kirby, Baylor University, Department of Mathematics, One Bear Place, Waco, TX, USA, robert_kirby@baylor.edu; Jorge Marchena-Menéndez, Baylor University, Department of Mathematics, One Bear Place, Waco, TX, USA, jorge_marchena1@baylor.edu.

2021. Manuscript submitted to ACM

Farrell 2014], although this code mainly aims to optimize the implementation of transient models by caching and reusing assembled matrices and solver data across timesteps.

In this paper, we present *Irksome*, a solution to the problem of obtaining effective time-discretizations. Rather than consider all possible kinds of time-stepping algorithms, we restrict ourselves to Runge–Kutta (RK) methods in this project. Runge–Kutta methods themselves are a vast family covering explicit and implicit methods of various orders of accuracy. When applications call for some kind of stability or conservation property, there is typically some suitable RK method available. Moreover, *Butcher tableaux* provide a unified description of RK methods and hence a unified entry point for automating their deployment in conjunction with UFL-based PDE descriptions. IMEX-type and partitioned Runge–Kutta methods that allow different methods for different terms in the equation can be of use for Hamiltonian systems, but such methods require additional UFL manipulation and we regard them as a future project. We also expect multi-step, generalized linear methods or other broad families of methods could admit similar deployment. An alternative approach would be to wrap PETSc’s TS package [Abhyankar et al. 2018]¹, providing access to these other kinds of methods currently available in PETSc. However, TS currently only provides a small, enumerated set of RK methods, and these do not include multi-stage implicit RK methods. A major motivation of our work is to study efficient algebraic solvers for the large, coupled algebraic systems such methods yield.

Our work follows the design principle of separating *mechanism* from *policy* [Lampson and Sturgis 1976]. That is, *Irksome* makes it straightforward to apply a particular RK method to a PDE, but does not comment on which method the user ought to select. In some ways, this mirrors the agnosticism of UFL and other domain-specific languages for spatial discretizations: UFL provides relatively broad access to classical, mixed, and discontinuous Galerkin methods without editorializing on their relative merits for a particular problem. Just as one may use UFL to implement both good and bad spatial discretizations, one may use *Irksome* to apply arbitrary Runge–Kutta schemes in time, and thereby compare their merits. It is up to the analyst to select an appropriate one and configure the linear solver accordingly. We provide some examples that may guide usage.

The discussion of optimal ‘policy’ regarding Runge–Kutta time-stepping of finite element discretizations of PDE is ongoing, and we hope to provide a useful tool in this respect. Despite their favorable accuracy and stability properties, higher-order fully implicit RK methods have not yet been extensively used in practical applications. Two reasons for this are typically given. First, as the algebraic system couples all RK stages, in a traditional finite element code the cost of implementation is quite high; the space and time discretization must be combined in the assembly process. Second, they have been considered impractically expensive, due to the size of the nonlinear systems to be solved. *Irksome* addresses both of these points: by automating the application of fully implicit RK methods, the first concern dissolves, and by building on the sophisticated solver infrastructure of Firedrake and PETSc [Kirby and Mitchell 2018], fast preconditioners can be developed and applied to address the second.

One response to the difficulties of implicit RK methods has been the development of ‘diagonally implicit’ Runge–Kutta methods (DIRKs). These can provide many favorable properties, but with a sequence of smaller algebraic systems for each stage. However, they are necessarily limited to low stage order, and in some cases effective preconditioners for fully implicit methods can be competitive in efficiency with DIRKs. Mardal, Nilssen, and Staff [Mardal et al. 2007] gave a rigorous analysis of certain block-diagonal preconditioners for fully implicit discretizations of parabolic PDE. Essentially, the cost of applying the preconditioner for an s stage method is the solution of s independent systems similar to that for a backward Euler step. When the outer iteration count is small, so is the relative cost of a fully

¹This has been recently done in <https://github.com/IvanYashchuk/firedrake-ts>.

implicit method compared to a DIRK. Huang *et al* [Huang *et al.* 2019] combine a diagonal preconditioner for backward Euler with the Jordan form of the Butcher matrix to precondition the higher-order system. Pazner and Persson [Pazner and Persson 2017] apply an earlier recommendation of Butcher for methods with invertible A matrix [Butcher 1976] to the compressible Navier–Stokes equations. In this technique, they change variables to render the stiffness part of Jacobian block diagonal and hence cheaper to apply and precondition. They find that with appropriate preconditioning, Radau IIA methods (which are L-stable and have relatively high stage order) can be made more efficient than DIRKs of comparable order.

In Section 2, we give a brief overview of Runge–Kutta methods. In particular, we are interested in methods given by a classical Butcher tableau, any of which can be encoded by a few arrays. In addition to a few general families of collocation methods chosen to highlight different stability features, we have also included a range of classical methods. In Section 3 we describe Irkosome, giving an overview of the library itself and some of the internal implementation. Several examples demonstrating various features are given in Section 4, and concluding thoughts are given in Section 5.

2 RUNGE–KUTTA METHODS

2.1 Overview

We first consider ordinary differential equations of the form

$$y'(t) + F(t, y) = 0, \quad (2.1)$$

where $F : (0, T] \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, and the solution $y : (0, T] \rightarrow \mathbb{R}^n$. The solution must also satisfy an initial condition

$$y(0) = y_0. \quad (2.2)$$

We will assume that standard conditions for the existence, uniqueness, and continuous dependence of solutions (e.g. via the Picard-Lindelöf theorem) hold for the discrete ODE.

Given the solution $y(t^n) \equiv y_n$ and some $t^{n+1} = t^n + \Delta t$, Runge–Kutta methods approximate $y(t^{n+1})$ by

$$y^{n+1} = y^n + \Delta t \sum_{i=1}^s b_i k_i, \quad (2.3)$$

where for all $1 \leq i \leq n$ the stages $k_i \in \mathbb{R}^n$ satisfy

$$k_i + F \left(t + c_i \Delta t, y + \Delta t \sum_{j=1}^s A_{ij} k_j \right) = 0. \quad (2.4)$$

The numbers A_{ij} , b_i , and c_i for $1 \leq i, j \leq n$ are in principle arbitrary but are chosen so that the resulting method has a given order of accuracy as well other desired properties (e.g. various notions of stability or symplecticity). They are typically organized in a *Butcher tableau*

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b} \end{array}, \quad (2.5)$$

where the vectors $\mathbf{b}, \mathbf{c} \in \mathbb{R}^s$ contain the entries b_i and c_i , respectively, and $A \in \mathbb{R}^{s \times s}$ contains the entries A_{ij} . If A is strictly lower triangular, then the method is explicit – each stage value can be computed in sequence without recourse to an algebraic system (modulo mass matrices in the variational context). Otherwise, the method is implicit. In the case of a fully implicit method (A being essentially dense), one must solve an $(ns) \times (ns)$ system of algebraic equations

to determine all the stage values simultaneously. When A is lower triangular but not strictly so (a diagonally implicit method or DIRK), one may solve s consecutive $n \times n$ algebraic systems for the stages.

More generally, discretizations of certain PDE (e.g. mixed formulations of the time-dependent Navier–Stokes) give rise to systems of (explicit) differential-algebraic equations

$$\begin{aligned} x'(t) + F(t, x, y) &= 0, \\ G(x, y) &= 0, \end{aligned} \tag{2.6}$$

or even more generally a fully implicit formulation

$$F(t, x, x') = 0. \tag{2.7}$$

In this paper, we investigate the application of RK methods to finite element spatial discretizations of (generally nonlinear) time-dependent PDE. For a finite-dimensional function space V_h , we consider variational evolution equations of the form of finding $u : (0, T] \rightarrow V_h$ such that

$$(u_t, v) + \mathcal{F}(t, u; v) = 0, \tag{2.8}$$

for all $0 < t \leq T$ and $v \in V_h$.

Here, we assume that v enters into \mathcal{F} linearly but make no particular assumptions about its dependence on t and u , other than that the discrete system is solvable. We do not assume that we have a particular spatial discretization – discontinuous Galerkin or other nonconforming methods fit into our framework as well as standard conforming ones.

It is useful to consider an even more general problem of finding $u : (0, T] \rightarrow V_h$ such that

$$\mathcal{G}(t, u, u_t; v) = 0, \tag{2.9}$$

for all $0 < t \leq T$ and $v \in V_h$. This general formulation allows some interesting cases (e.g. Sobolev equations) not covered in (2.8), and working with this abstract form makes our UFL manipulation more straightforward.

The Runge–Kutta methods we consider have a range of different kinds of stability properties appropriate for various PDE examples we consider later. We refer the reader to [Hairer et al. 2006; Wanner and Hairer 1996] for an in-depth discussion of these various properties. The RK methods we consider are A -stable, meaning that if applied to the test equation $y' = \lambda y$ with $\text{Re}(\lambda) < 0$, the computed solution tends to zero at infinity for all positive time-steps. While the famous Dahlquist Barrier Theorem [Dahlquist 1963] says that A -stable multi-step methods can be at most second order accurate, A -stable RK methods of all orders are known. A -stability potentially allows large time steps to be chosen, provided sufficient accuracy is obtained with them. For very stiff equations, a stronger notion of stability than A -stability is desirable. So-called L -stability requires that the stability function vanish at infinity. Additionally, some ODE systems possess a monotonicity property that if $y' = f(t, y)$ and $z' = f(t, z)$ are solutions with different initial conditions, then $\|y(t_2) - z(t_2)\| \leq \|y(t_1) - z(t_1)\|$ for $t_1 \leq t_2$. B -stable numerical methods [Wanner and Hairer 1996] preserve a discrete analog of this property.

Finally, some ODE possess one or more conserved quantities, and certain RK methods excel in preserving this in the discretization. For problems with a Hamiltonian structure, *symplectic* methods conserve the Hamiltonian up to a perturbation over exponentially long time scales (the result is stronger for linear problems). Some methods are additionally known to conserve linear or quadratic invariants, which makes them highly attractive for problems with these features. We demonstrate such properties for the linear wave equation and the nonlinear Benjamin–Bona–Mahony and Gross–Pitaevskii equations.

2.2 PDE examples

To fix ideas, we now consider a range of PDE to serve as motivating examples. These examples are chosen to explore two different dimensions of our work. First, they cover ODE, DAE, and implicit/Sobolev PDE. Second, these examples call for different kinds of stability properties, highlighting the need for different methods.

2.2.1 The heat equation. A model problem for time stepping finite element discretizations is the heat equation

$$(u_t, v) + (\nabla u, \nabla v) = (f, v), \quad (2.10)$$

posed on some domain $\Omega \subset \mathbb{R}^d$ with $d \in \{1, 2, 3\}$, together with Dirichlet boundary conditions $u|_{\partial\Omega} = g(t, \cdot)$ and data $f : (0, T] \times \Omega \rightarrow \mathbb{R}$. We let V_h consist of standard continuous piecewise polynomials defined over a triangulation of Ω .

Backward Euler is a very common method, highly stable but only first-order accurate, for the heat equation. Given u^n , the approximation to the solution u at time t_n , we define u^{n+1} as the solution to the variational problem

$$\left(\frac{u^{n+1} - u^n}{\Delta t}, v \right) + (\nabla u^{n+1}, \nabla v) - (f(t^{n+1}, \cdot), v) = 0, \quad (2.11)$$

for all $v \in V_h$. Since u^{n+1} is the unknown value and u^n is data for this problem, we can rearrange this to obtain

$$(u^{n+1}, v) + \Delta t (\nabla u^{n+1}, \nabla v) = (u^n, v) + \Delta t (f(t^{n+1}, \cdot), v). \quad (2.12)$$

Although backward Euler is just a Runge–Kutta method with Butcher tableau

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}, \quad (2.13)$$

one typically poses the problem for u^{n+1} , as we have done in (2.11) or (2.12), instead of the lone stage k_1 . If we apply a generic s -stage RK method to (2.10), we pose a variational problem for the s stages. We seek $\{k_i\}_{i=1}^s \subset V_h$ such that

$$(k_i, v_i) + \left(\nabla \left(u^n + \Delta t \sum_{j=1}^s a_{ij} k_j \right), \nabla v_i \right) - (f(t + c_i \Delta t, \cdot), v_i) = 0, \quad (2.14)$$

for each $v_i \in V_h$, and then find u^{n+1} by

$$u^{n+1} = u^n + \Delta t \sum_{i=1}^s b_i k_i. \quad (2.15)$$

In the case of backward Euler, the variational equation for the stage k_1 is similar to (2.11). If we substitute in (2.13), we find

$$(k_1, v_1) + (\nabla (u^n + \Delta t k_1), \nabla v_1) - (f(t + \Delta t, \cdot), v_1) = 0. \quad (2.16)$$

Now, we can rearrange this to give an equation for k_1 :

$$(k_1, v_1) + \Delta t (\nabla k_1, \nabla v_1) = (f(t + \Delta t, \cdot), v_1) - (\nabla u^n, \nabla v_1). \quad (2.17)$$

Although the right-hand side is different, the quantity k_1 here can be put into 1–1 correspondence with u^{n+1} from (2.11) by $k_1 \leftrightarrow \frac{u^{n+1} - u^n}{\Delta t}$. Importantly, we note the equality of the bilinear form defining u^{n+1} in (2.12) to that defining k_1 in (2.17).

While the Runge–Kutta formulation (2.14) is quite a bit different than (2.11), we can programmatically obtain it from (2.10) by, for each stage, substituting k_i in for $\frac{\partial u_i}{\partial t}$, $u^n + \Delta t \sum_{j=1}^s a_{ij} k_j$ for u , v_i for v , and $t + c_i \Delta t$ for t . This simple insight forms the basis of the implementation, which we discuss in section 3.

Strongly-enforced boundary conditions for the semidiscrete problem must be converted to boundary conditions for each Runge–Kutta stage. The stage k_i approximates the time derivative u_t at time $t^n + c_i \Delta t$, so if we pose a boundary condition of the form

$$u|_\Gamma = g(t, \cdot),$$

for some boundary segment $\Gamma \subset \partial\Omega$, we impose the corresponding stage boundary condition

$$(k_i)|_\Gamma = \frac{\partial g}{\partial t}(t^n + c_i \Delta t, \cdot). \quad (2.18)$$

We note that weakly-enforced boundary conditions (such as Neumann and Robin for standard Galerkin methods) require no special treatment, just the evaluation of any time-dependent data at the correct stage times.

The discrete heat equation is a prototypical stiff system. Explicit time-stepping methods require a time step with $\Delta t = O(h^2)$, where h is the mesh size. A -stability allows us to put $\Delta t = O(h)$, with a constant possibly larger than 1. Also, because the stiffness increases under mesh refinement, L -stability can be attractive. Hence, we would expect Gauss–Legendre methods to be passable but perhaps to obtain better results from RadauIIA and LobattoIIIC families for a given order. While the heat equation does possess a contractive property, B -stability does not seem as important in practice for these problems.

Generalizing the class of problems we consider, the mixed form of the heat equation gives rise to a system of differential algebraic equations (DAE). Introducing the new variable $\sigma = -\nabla u$ in the heat equation, we obtain the system of PDE

$$\begin{aligned} u_t + \nabla \cdot \sigma - f &= 0, \\ \sigma + \nabla u &= 0. \end{aligned} \quad (2.19)$$

To discretize this, we take $W_h^1 \subset L^2$ as the space of discontinuous piecewise polynomials of degree k over a triangulation of Ω and $W_h^2 \subset H(\text{div})$ a suitable mixed approximating space and define $V_h = W_h^1 \times W_h^2$. We then seek $(u, \sigma) : (0, T] \rightarrow V_h$ such that

$$\begin{aligned} (u_t, v) + (\nabla \cdot \sigma, v) - (f, v) &= 0, \\ (\sigma, w) - (u, \nabla \cdot w) &= 0, \end{aligned} \quad (2.20)$$

for all $(v, w) \in V_h$. Since only u_t appears in the equation and not σ_t , we have a differential-algebraic system rather than just a system of ODE. Still, we can apply a generic RK method to (2.20) to obtain a coupled system of variational problems for all s stages. For each $1 \leq i \leq s$, we seek $(k_i^u, k_i^\sigma) \in V_h$ such that

$$\begin{aligned} (k_i^u, v_i) + \left(\nabla \cdot \left(\sigma^n + \Delta t \sum_{j=1}^s a_{ij} k_j^\sigma \right), v_i \right) - (f(t_n + c_i \Delta t, \cdot), v_i) &= 0, \\ \left(\left(\sigma^n + \Delta t \sum_{j=1}^s a_{ij} k_j^\sigma \right), w_i \right) - \left(\left(u^n + \Delta t \sum_{j=1}^s a_{ij} k_j^u \right), \nabla \cdot w_i \right) &= 0, \end{aligned} \quad (2.21)$$

for all $(v_i, w_i) \in V_h$.

In some sense, DAE are “infinitely stiff”, and so we expect A - and L -stability to be quite important for these problems. We therefore expect RadauIIA or LobattoIIIC methods to outperform Gauss–Legendre.

2.2.2 *The wave equation.* We can write the wave equation $u_{tt} - \Delta u = 0$ as a first-order variational system

$$\begin{aligned} (u_t, v) + (\nabla \cdot \sigma, v) &= 0, \\ (\sigma_t, w) - (u, \nabla \cdot w) &= 0. \end{aligned} \tag{2.22}$$

Following [Geveci 1988; Kirby and Kieu 2015], we take the flux variable $\sigma \in H(\text{div})$ and $u \in L^2$. As above, we take W_h^1 as the space of discontinuous piecewise polynomials of degree k over a triangulation of Ω and $W_h^2 \subset H(\text{div})$ a suitable mixed approximating space and define $V_h = W_h^1 \times W_h^2$.

The resulting ODE tend to be far less stiff than those for heat equation. Energy conservation rather than stiffness tends to be the central issue for time stepping. Standard integrators like forward or backward Euler dramatically fail to conserve energy, and one typically requires some kind of symplectic integrator. Kirby and Kieu [2015] analyzed a first-order symplectic Euler time-stepping scheme for this problem, and Kernell and Kirby [2020] studied preconditioners for a Crank–Nicolson time discretization of a slightly more general equation.

Applying a generic Runge–Kutta method to (2.22) gives

$$\begin{aligned} (k_i^u, v_i) + \left(\nabla \cdot \left(\sigma^n + \Delta t \sum_{j=1}^s a_{ij} k_j^\sigma \right), v_i \right) - (f(t_n + c_i \Delta t, \cdot), v_i) &= 0, \\ (k_i^\sigma, w_i) - \left(\left(u^n + \Delta t \sum_{j=1}^s a_{ij} k_j^u \right), \nabla \cdot w_i \right) &= 0. \end{aligned} \tag{2.23}$$

Some Runge–Kutta families, such as Gauss–Legendre methods, provide high-order, A -stable, and symplectic methods that preserve system energy for linear problems (and nearly so for nonlinear ones) [Hairer et al. 2006]. The damping feature that makes RadauIIA and LobattoIIIC quite suitable for the heat equation turns out to be a major weakness for the wave equation. Despite the different RK method to be preferred, we note that the discrete system (2.23) has quite a similar structure to (2.21).

2.2.3 *Nonlinear examples.* As an example of a PDE system giving rise to a nonlinear DAE system, we consider the incompressible Navier–Stokes equations

$$\begin{aligned} u_t - \nu \Delta u + u \cdot \nabla u + \nabla p &= 0, \\ \nabla \cdot u &= 0, \end{aligned} \tag{2.24}$$

where u is the vector-valued fluid velocity and p the pressure, and the parameter ν is the kinematic viscosity. Among the many issues these equations present, one must choose suitably compatible discrete spaces for velocity and pressure to ensure stability.

We also consider the Benjamin–Bona–Mahony [Benjamin et al. 1972] (BBM) equation

$$u_t + u_x + uu_x - u_{txx} = 0, \tag{2.25}$$

typically posed on $\Omega = \mathbb{R}$ or a periodic domain. It has multivariate extensions and is also similar to more complex models arising in magma dynamics [Simpson and Spiegelman 2011]. For our purposes, it is a nonlinear Sobolev-type equation, with spatial derivatives acting on time derivatives in the u_{txx} term.

The BBM equation has a Hamiltonian structure and three polynomial invariants:

$$I_1 = \int u \, dx, \quad (2.26)$$

$$I_2 = \int u^2 + (u_x)^2 \, dx, \quad (2.27)$$

$$I_3 = \int (u_x)^2 + \frac{1}{3}u^3 \, dx. \quad (2.28)$$

Each of these quantities remains constant over time. Note that I_1 is a linear invariant, I_2 quadratic, and I_3 cubic. The BBM equation also supports solitary (but non-soliton) wave solutions.

As a final nonlinear example, we consider the Gross–Pitaevskii [Gross 1961; Pitaevskii 1961] (GP) equation in a parabolic trap

$$i\psi_t = -\frac{1}{2}\nabla^2\psi + |\psi|^2\psi + \frac{1}{2}|\mathbf{x}|^2\psi, \quad (2.29)$$

where $\psi : (0, T] \times \Omega \rightarrow \mathbb{C}$ is a complex-valued wavefunction. This equation describes the dynamics of Bose–Einstein condensates, a phase of matter where a gas of bosons condenses into the same quantum state. The equation is of substantial interest in physics, and supports nonlinear waves such as solitons and vortices [Kevrekidis et al. 2015]. The invariants of this equation that we consider are the number of atoms and the energy:

$$J_1 = \int_{\Omega} |\psi|^2 \, dx, \quad (2.30)$$

$$J_2 = \frac{1}{4} \int |\nabla\psi|^2 + |\mathbf{x}|^2|\psi|^2 + |\psi|^4 \, dx. \quad (2.31)$$

The number of atoms J_1 is a quadratic functional, while the energy J_2 is quartic. A second-order single-step single-stage scheme that conserves both invariants to machine precision has been proposed by [Delfour et al. 1981].

2.3 Algebraic systems

A long-standing critique of higher-order implicit RK methods, and argument for DIRKs, is the size and complexity of the algebraic systems required to be solved at each time step. We now discuss some of these issues and attempt to address them in the context of the heat equation with a 3-stage method. We define M and K to be the standard finite element mass and stiffness matrices, with

$$M_{ij} = \int_{\Omega} \psi_i \psi_j \, dx, \quad K_{ij} = \int_{\Omega} \nabla \psi_i \cdot \nabla \psi_j \, dx \quad (2.32)$$

where $\{\psi_i\}_{i=1}^{\dim V_h}$ is a finite element basis. The variational problem (2.14) gives rise to a block algebraic system of the form

$$\left(\begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{bmatrix} + \Delta t \begin{bmatrix} a_{11}K & a_{12}K & a_{13}K \\ a_{21}K & a_{22}K & a_{23}K \\ a_{31}K & a_{32}K & a_{33}K \end{bmatrix} \right) \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \mathbf{k}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \end{bmatrix}, \quad (2.33)$$

or equivalently (and true for general s -stage methods for linear problems)

$$(I \otimes M + \Delta t A \otimes K) \mathbf{k} = \mathbf{f}. \quad (2.34)$$

The Jacobian for nonlinear problems will similarly couple together all of the RK stages, although without the Kronecker product structure in the stiffness matrix (unless a modified Newton scheme is used, e.g. [Bickart 1977; Butcher 1976]).

Following [Mardal et al. 2007], the block diagonal of this system makes an excellent preconditioner, at least in the case of parabolic problems such as the heat equation:

$$P = \begin{bmatrix} M + a_{11}\Delta t K & 0 & 0 \\ 0 & M + a_{22}\Delta t K & 0 \\ 0 & 0 & M + a_{33}\Delta t K \end{bmatrix}. \quad (2.35)$$

The diagonal blocks are the same as matrices obtained by discretizing backward Euler with an altered time step. More generally, one may also consider a block triangular preconditioner [Staff et al. 2006] such as:

$$P = \begin{bmatrix} M + a_{11}\Delta t K & 0 & 0 \\ a_{21}K & M + a_{22}\Delta t K & 0 \\ a_{31}K & a_{32}K & M + a_{33}\Delta t K \end{bmatrix}. \quad (2.36)$$

Applying this preconditioner still only requires linear solves with the diagonal blocks, and typically leads to somewhat faster convergence than (2.35). Some form of multigrid iteration (either algebraic or geometric) will provide an excellent preconditioner for these blocks. Instead of taking the Kronecker product of a triangular part of A with K , it is also possible to use a triangular factor from the LDU decomposition of A , and this seems to give additional reductions in iteration count with comparable cost-per-iteration [Rana et al. 2020].

We note that if a DIRK method is used so that A is lower-triangular, (2.33) becomes

$$\left(\begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{bmatrix} + \Delta t \begin{bmatrix} a_{11}K & 0 & 0 \\ a_{21}K & a_{22}K & 0 \\ a_{31}K & a_{32}K & a_{33}K \end{bmatrix} \right) \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \quad (2.37)$$

In this case, the triangular structure means one can proceed by forward substitution, solving in turn for each of the stages. Moreover, an *explicit* method (typically not recommended for the heat equation) would have A strictly lower triangular and the linear system

$$\left(\begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & M \end{bmatrix} + \Delta t \begin{bmatrix} 0 & 0 & 0 \\ a_{21}K & 0 & 0 \\ a_{31}K & a_{32}K & 0 \end{bmatrix} \right) \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}, \quad (2.38)$$

so that we only must solve the mass matrix once per stage (and do mat-vec and axpy-type operations) to obtain all of the stages. One can also apply the block lower-triangular preconditioner (via PETSc’s `FIELDSPLIT` [Brown et al. 2012]) for DIRK and explicit methods, which exactly solves the linear systems in a single outer iteration, provided that the diagonal blocks are solved accurately. Hence, the range of RK methods enabled in Irkosome composes with the existing Firedrake code stack to enable efficient implementations.

Rather than splitting the stages as with the block diagonal or triangular preconditioners, it is also possible to apply a monolithic multigrid scheme using some kind of blockwise smoothing. The theory of such methods is quite undeveloped, but our techniques seem robust in practice. We include some empirical results in this direction for both the heat equation in Section 4.2 and the Gross–Pitaevskii equation in Section 4.4.3.

We remark that the Firedrake solver infrastructure makes it possible to apply the (rather large) operator in a matrix-free fashion, saving large amounts of memory by computing the action of the Jacobian on a vector rather than first assembling a sparse matrix and performing a matrix-vector product. Firedrake is not yet able to detect or exploit any Kronecker product structure, if available. Preconditioners that work only with matrix actions (such as geometric

multigrid with rediscritization) require no further modification, while one may use a `firedrake.AssembledPC` preconditioner to force assembly of particular diagonal blocks if the actual sparse matrix values are needed (such as for algebraic multigrid). Such techniques approaches are further discussed in [Kirby and Mitchell 2018].

3 IRKSOME

Irksome is a Python library for manipulating UFL for semidiscrete variational forms and strongly-enforced boundary conditions into UFL for fully discrete Runge–Kutta methods. It is available on GitHub under the Firedrake project umbrella at <https://github.com/firedrakeproject/Irksome>, and can be installed as part of Firedrake by adding the `--install irksome` option to the Firedrake installation script.

As exploiting the PDE structure of the algebraic system is key for obtaining efficient solvers, it is crucial that we maintain this symbolic information in our implementation. Consequently, Irksome’s approach based on UFL manipulation seems to afford richer opportunities than a black-box RK library could. As concrete examples, maintaining the symbolic structure enables the straightforward implementation of monolithic multigrid and advanced field-split (Schur complement) preconditioners to the implicit RK system, and makes possible efficient vectorized matrix-free implementations [Sun et al. 2020].

Irksome’s main actions are performed by a function `getForm` that takes UFL for the time-dependent variational form, a Butcher tableau, the current time and time step (both stored as UFL Constant objects), and a UFL Coefficient. Irksome also provides a collection of Runge–Kutta methods in a separate module (see Subsection 3.2). The function `getForm` returns UFL for the coupled, multi-stage method and boundary conditions. We also provide a convenience class, `TimeStepper` that handles interactions with `getForm`, boundary conditions, and variational solvers and advances the solution forward in time. This class and the module containing Butcher tableaux are the main user entry points for Irksome.

3.1 UFL manipulation

To illustrate what Irksome automates, we consider the semidiscrete homogeneous heat equation, which can be written in UFL (with our support of a `Dt` operator) as

$$F = \text{inner}(\text{Dt}(u), v) * dx + \text{inner}(\text{grad}(u), \text{grad}(v)) * dx$$

Typically, we would express backward Euler for the heat equation in UFL as

$$F = \text{inner}((\text{unew} - u)/dt, v) * dx + \text{inner}(\text{grad}(\text{unew}), \text{grad}(v)) * dx$$

However, Runge–Kutta methods require specifying the variational form the update stages satisfy rather than the value at the next time step. To fix ideas, consider the two-stage LobattoIIIC method given by the Butcher tableau

$$\begin{array}{c|cc} 0 & 1/2 & -1/2 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array} \quad (3.1)$$

Following (2.14), we could write the variational form for the two stages as in Listing 1, and solve the variational problem $F=0$ for the unknown k_i , and use it to update the solution.

Using a different Butcher tableau leads to a similar variational problem, simply replacing the entries of A . More generally, we also must utilize c to evaluate explicitly time-dependent expressions (say, in material properties or

```

# u is a given Function in V containing solution at time n
# dt is a Constant holding the current time step value
Vbig = V * V
k = Function(Vbig)
k0, k1 = split(k)
v0, v1 = TestFunctions(Vbig)
u0 = u + Constant(0.5) * dt * k0 + Constant(-0.5) * dt * k1
u1 = u + Constant(0.5) * dt * k0 + Constant(0.5) * dt * k1

F = (inner(k0, v0) * dx + inner(k1, v1) * dx +
     inner(grad(u0), grad(v0)) * dx + inner(grad(u1), grad(v1)) * dx)

```

Listing 1. Sample UFL for a two-stage Runge–Kutta discretization of the heat equation.

forcing terms). Because the transformation from the semidiscrete form of F to its Runge–Kutta variational form is quite mechanical, it can be mechanized.

In the case of a single, scalar-valued equation, these transformations can be encoded in a straightforward way. One can build a space V_{big} that is an s -way Cartesian product of the underlying function space and create a `Function` called k for the stages and a `TestFunction` over the stages as well. For each stage, one performs a small set of substitutions. First, the test function in the semidiscrete form is replaced by the relevant component of the test function over the product space. The time derivative $Dt(u)$ is replaced by the relevant piece of k as well. The replacement for u is somewhat more complicated – at stage i one replaces u with u plus dt times a row of the Butcher matrix A contracted with the stage variables. For mixed systems and vector-valued fields, the symbolic substitutions become somewhat more involved, as one must map from the unknown fields and their time derivatives into offsets into k appropriately.

Also, even if the formulae for substitution are fairly clear, the mechanics of performing the substitutions require some care. Suppose we have a vector-valued unknown u and that $u[i]$ appears in the semidiscrete bilinear form. The existing `ufl.replace` is based on a post-order multifunction — it takes an expression and a dictionary of replacements to perform and then replaces a node in the expression graph if it matches something in the dictionary. Hence, when $u[i]$ appears in an expression, the u will be found first and then replaced with its evaluation at a stage, indexed with component i . This will point to the wrong piece of the stage variable k . Similar difficulties can occur with time derivatives. To address this, we wrote a modified replacement function based on *pre-order* traversal. If an expression matches something in the set of replacements to be performed, the replacement gets made, and otherwise recursion occurs on the expression’s children. Hence, the match for $u[i]$ is found and the correct substitution is performed. While this form of traversal/substitution is less efficient than post-order, the overhead of UFL manipulation is negligible in almost all contexts.

Similarly, substitution is performed on boundary condition values. Some care must be taken in both the variational form and boundary conditions to handle mixed problems where the underlying function space is itself the Cartesian product of multiple spaces. Also, in our current implementation, we assume that time derivatives are applied only to unknown fields and not to algebraic combinations of expressions and fields (e.g. one must write $2*t*u + t**2 * Dt(u)$ rather than $Dt(t**2 * u)$ and $2*Dt(u)*u$ rather than $Dt(u**2)$). This does not seem to be a major limitation, but could be relaxed by first applying a transformation expanding such derivatives before applying substitutions.

3.2 Currently available RK methods

The universe of known Runge–Kutta methods is vast. While any RK method can be implemented by passing the appropriate Butcher tableau, Irksome provides implementations of many familiar classical methods, with a particular focus on fully implicit collocation-type methods. These typically have an adjustable parameter for the order, much like the function space constructors in FEniCS and Firedrake.

Among collocation-type Runge–Kutta methods, we have implemented Gauss–Legendre (which includes implicit midpoint), LobattoIIIA (which includes Crank–Nicolson), and RadauIIA (which includes backward Euler) methods. We use FIAT [Kirby 2004] to obtain the interpolating nodes for c , and then explicit formulae (evaluated by numerical integration) give the entries of A and b in the Butcher tableau. Since FIAT can compute the quadrature points to arbitrary degree, we similarly have arbitrary-order implementation of these methods. These methods are all A -stable. The Gauss–Legendre methods are B -stable and symplectic, but not L -stable, while the RadauIIA methods are L -stable. We have also implemented general-order LobattoIIIC methods, which have the same b and c arrays as LobattoIIIA but a different A matrix. They are not quite standard collocation methods, but are L -stable and B -stable as well as A -stable and frequently recommended for stiff problems.

In addition to these methods, we provide Butcher tableaux for a range of other classical methods (forward Euler, an SSP method [Gottlieb et al. 2001], explicit midpoint/trapezoid rules, the classical fourth-order method, and some L -stable DIRKs [Alexander 1977]).

4 EXAMPLES AND NUMERICAL RESULTS

4.1 The heat equation

In this section, we confirm the accuracy of our generated methods for the heat equation via the method of manufactured solutions. We let $\Omega = (0, 1) \times (0, 1)$ be the unit square and select the forcing function f and Dirichlet boundary conditions such that the true solution is $u(x, y, t) = e^{-t} \sin(\pi x) \cos(\pi y)$.

For the primal form of this equation, we divide Ω into an $N \times N$ mesh of squares for $N = 8, 16, 32, 64, 128$ and take V_h to be the space of cubic serendipity finite elements. For smooth enough solutions, these lead to fourth order L^2 error and third order H^1 error, so we focus on time-stepping methods of order at least three – Gauss–Legendre(2) and LobattoIIIC(3) (both of order 4) and two- and three-stage RadauIIA methods of order 3 and 5, respectively.

The numerical results illustrate the effect of *order reduction* [Wanner and Hairer 1996]. For stiff problems, collocation-type methods do not yield the full nominal order of accuracy (e.g. order $2s$ for an s -stage Gauss–Legendre method) but reduce to the maximum truncation error per-stage (e.g. s for an s -stage Gauss–Legendre method). For time steps large enough for temporal error to be significant relative to spatial error, we see a reduction in the observed accuracy. For example, Figure 1a shows fourth order accuracy in L^2 using the two-stage Gauss–Legendre method when the time step is very small, but this degenerates to stiff order of two. A similar reduction of the H^1 convergence rate below third order is observed for larger time steps. The L^2 error using LobattoIIIC(3) also reduces for larger time steps, but the H^1 error stays at order three (which is both the stage order and the spatial order in this norm). The RadauIIA(2) method with stiff order two has order reduction in L^2 norm, but none is observed in H^1 (presumably the time truncation error is small enough not to affect the overall convergence order). The RadauIIA(3) method has stage order three and so does not affect the H^1 error, and its time truncation error is small enough that, for the time steps we consider, we observe full accuracy in L^2 as well.

We also consider the mixed form of the heat equation, now dividing the $N \times N$ mesh into right triangles and using third-order Raviart–Thomas elements for V_h with discontinuous polynomials of degree three for W_h . Hence, both variables u_h and σ_h should be third-order accurate in space in the L^2 norm, as should $\nabla \cdot \sigma_h$. In Figure 2a, we see order reduction for a two-stage Gauss–Legendre method, although no order reduction for the three-stage RadauIIA method is observed in Figure 2b. From this, we see that the extra formal order of accuracy for Gauss–Lobatto methods does not materialize unless one also takes a very small time step. We also obtained satisfactory results with somewhat

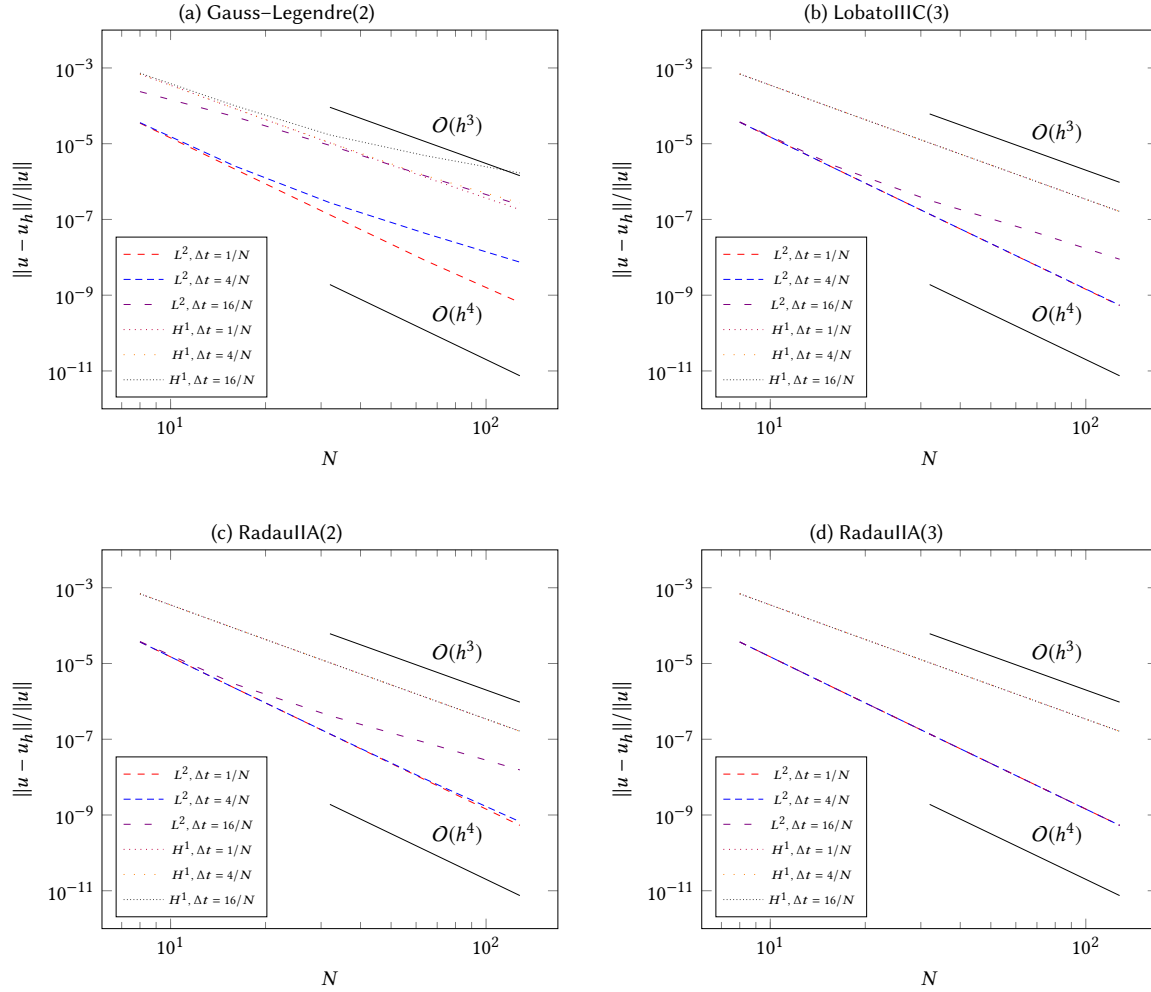


Fig. 1. Relative L^2 and H^1 errors at $t = 2$ on an $N \times N$ mesh using various time stepping schemes and CFL numbers.

lower accuracy with the two-stage RadauIIA method (but at a lower cost) and the three-stage LobattoIIIC method (at a comparable cost). Note that the $H(\text{div})$ error for σ and L^2 error for u are similar but not identical for the RadauIIA method. This approximation of the flux variable with comparable accuracy to the potential is seen as a major advantage of mixed methods.

4.2 Preconditioning of the heat equation

We now consider preconditioning the algebraic system (2.33) obtained by applying RadauIIA methods to a Galerkin discretization of the heat equation. In this section, all timings reported are performed on a MacBook Pro running macOS

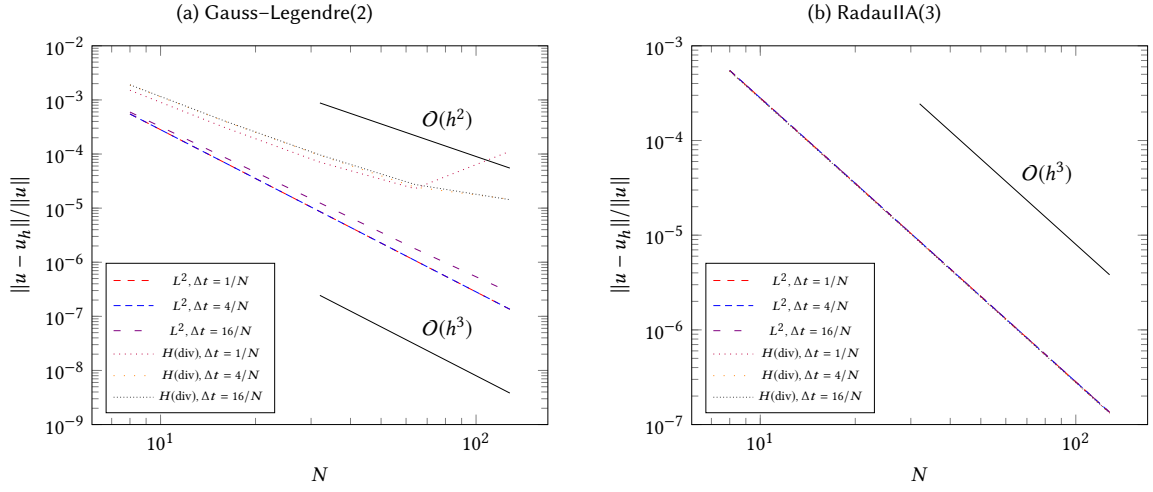


Fig. 2. Relative L^2 and $H(\text{div})$ errors at $t = 2$ for the mixed heat equation on an $N \times N$ mesh divided into right triangles using various time stepping schemes and CFL numbers.

10.15.6 (Catalina) with 32GB RAM and a 2.6GHz 6-Core Intel Core i7 processor. All timings are performed on single core for ease of comparison.

On one hand, we consider the preconditioners (2.35) and (2.36), using a multigrid algorithm for each diagonal block. On the other hand, we propose a monolithic multigrid method with a smoother that addresses the coupling between stages. These methods coincide in the case of the single-stage backward Euler method.

For the single-stage method (or each diagonal block in the split preconditioners), we consider a geometric multigrid method. On each grid in the hierarchy, we use Chebyshev-accelerated *vertex-star iteration*, a block-Jacobi method where the blocks are given by the degrees of freedom located strictly within the patch of cells surrounding each vertex (i.e. degrees of freedom on the boundary of each patch are excluded) [Farrell et al. 2020; Pavarino 1993; Schöberl et al. 2008]. In the case of P^1 or Q^1 discretizations, this reduces to point Jacobi smoothing. As the degree increases, one solves small systems over the patches around each vertex. For symmetric coercive problems, this is known to give condition numbers independent of the polynomial degree used and hence will perform well on the blocks of (2.35) and (2.36).

We can also attempt the same strategy applied directly to the fully coupled (and hence nonsymmetric) system without first introducing a block approximation. In this case, we have a more expensive smoother – with s stages, one has s times as many local degrees of freedom in each vertex patch as in the single-stage case. Van Lent and Vandewalle [2005] considered a block Jacobi smoother coupling s degrees of freedom at each spatial grid point for a centred finite difference discretization; the vertex-star iteration provides a natural generalization of this to higher-order discretizations. Because of the more expensive vertex-star smoother, the resulting multigrid algorithm must yield convergence in few iterations in order to be competitive.

Firedrake has a powerful interface to the PETSc multigrid algorithms [Lange et al. 2016; Mitchell and Müller 2016], and it also provides PETSc-accessible Schwarz methods via the PatchPC package [Farrell et al. 2020]. In Figure 2, we show suitable options for configuring a multigrid preconditioner using an additive Schwarz smoother. The type of

```

mg_params = {"levels":
    {
        "ksp_type": "chebyshev",
        "ksp_norm_type": "unpreconditioned",
        "pc_type": "python",
        "pc_python_type": "firedrake.PatchPC",
        "patch":
            {
                "pc_patch_save_operators": True,
                "pc_patch_partition_of_unity": False,
                "pc_patch_construct_type": "star",
                "pc_patch_construct_dim": 0,
                "pc_patch_sub_mat_type": "seqdense",
                "pc_patch_dense_inverse": True,
                "pc_patch_precompute_element_tensors": None,
                "sub_ksp_type": "preonly",
                "sub_pc_type": "lu"
            }
    },
    "coarse": {
        "pc_type": "lu",
        "pc_factor_mat_solver_type": "mumps"}
}

```

Listing 2. Options to configure a multigrid preconditioner to use patchwise additive Schwarz smoothing and a direct solver on the coarse grid.

```

split_params = {"snes_type": "ksponly",
    "ksp_type": "fgmres",
    "ksp_monitor_true_residual": None,
    "pc_type": "fieldsplit",
    "pc_fieldsplit_type": "additive"}

per_field = {"ksp_type": "preonly",
    "pc_type": "mg",
    "mg": mg_params}

for s in range(butcher_tableau.num_stages):
    split_params[f"fieldsplit_{s}"] = per_field

```

Listing 3. Options for solving (2.33) with a Krylov method preconditioned with (2.35). The inverse of the diagonal blocks is approximated with a single multigrid V-cycle as given in Listing 2, but other choices are possible.

```

monolithic_params = {"mat_type": "aij",
    "snes_type": "ksponly",
    "ksp_type": "fgmres",
    "ksp_monitor_true_residual": None,
    "pc_type": "mg",
    "mg": mg_params}

```

Listing 4. Options for solving (2.33) with a Krylov method preconditioned by the monolithic multigrid method described in (2). Here, the smoother couples together the stages in each vertex patch.

multigrid cycle, not specified in the code listing, will default to a V-cycle. In Figure 3, we subsequently show how to obtain the block diagonal preconditioner (2.35) and apply that multigrid preconditioner to each internal block. The block triangular preconditioner (2.36) is simply obtained by changing the "pc_fieldsplit_type" parameter from "additive" to "multiplicative". Also, Figure 4 shows how we can apply the monolithic, stage-coupled multigrid algorithm directly to the algebraic system at each time step. In all cases the flexible GMRES method [Saad 1993] is used as outer Krylov solver. We use the default PETSc Krylov convergence tolerances (a relative tolerance on the Euclidean norm of 10^{-5} , and an absolute tolerance of 10^{-50} , whichever comes first).

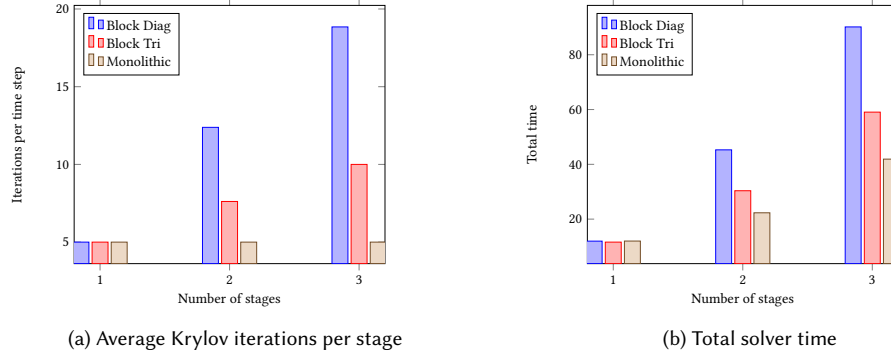


Fig. 3. Solver performance for the heat equation with Q^2 elements on a 128×128 mesh using RadauIIA time stepping with various numbers of stages. We compare the block diagonal (2.35) and block triangular (2.36) preconditioners to a monolithic multigrid approach.

For our experiment, we integrate the heat equation from $t = 0$ to $t = 1$ on a 128×128 mesh of squares. We use Q^2 elements in space, RadauIIA(s) methods in time with $s = 1, 2, 3$, and a time step of $\Delta t = 0.078125$. (Recall that s refers to the number of stages and not the formal order of the method). In Figure 3, we report the total solver time over all time steps and the average flexible GMRES iteration count per time step using the preconditioners we have discussed.

We first consider the block diagonal preconditioner (2.35), in the left columns of Figure 3. RadauIIA(2) is approximately four times as expensive as RadauIIA(1) (backward Euler). Moreover, RadauIIA(3) is about twice again as expensive. Since the number of Krylov iterations increases more slowly for the block triangular preconditioner (2.36) (middle columns in Figure 3), the increase in run-time over the single-stage method is somewhat smaller.

Results for the monolithic multigrid algorithm are shown in the right-hand columns of Figure 3. Unlike the block preconditioners, the number of Krylov iterations required per time step remains essentially constant as a function of the number of stages. The cost incurred by inter-stage coupling in the smoother is more than offset by the low iteration counts – the 2-stage method gives only about twice the run-time as the single stage method and the 3-stage method only costs about four times as much. Although no theory is yet available to explain the performance of the monolithic preconditioner, it seems worthy of further consideration.

Having seen the potential efficiency of our monolithic scheme compared to the split scheme, we also offer a brief study of robustness with respect to the mesh size and time step. Here, we solve the linear system associated with a single RadauIIA time step with 1, 2, or 3 stages. We use a range of $N \times N$ meshes with $N = 16, 32, 64, 128$. For each N , we consider time steps $\Delta t = c/N$ for $c = 1, 4, 16$. In each case, we found only 4 or 5 Krylov iterations preconditioned with the monolithic multigrid were required to obtain convergence. We report the results for the 3-stage method in Table 1; the 1- and 2- stage methods gave very similar results.

This robustness of our preconditioner with respect to the time step means we can hope to obtain a considerable speed-up with multi-stage methods. We conclude this section with a brief example to illustrate this. We solve the heat equation on a 32×32 mesh and choose the forcing and boundary data to agree with a given analytic solution. We integrate the system from until time $T = 1$ using a range of time steps and RadauIIA methods with 1, 2, and 3 stages, using our monolithic preconditioner at each time step. If we perform the temporal integration with decreasing time steps until the L^2 spatial error at $T = 1$ stops decreasing (indicating that spatial error dominates), we find the error is about 2.8×10^{-4} . Table 2 shows the run-time and final error obtained for a range of time steps using 1, 2, and 3 stage

	Δt		
N	$\frac{1}{N}$	$\frac{4}{N}$	$\frac{16}{N}$
16	4	5	5
32	5	5	5
64	5	5	5
128	5	5	5

Table 1. Number of outer Krylov iterations to solve (2.33) for $s = 3$ using the monolithic multigrid options in Figure 4, as a function of mesh size and time step. We use a Q^2 spatial discretization on an $N \times N$ mesh and let $\Delta t = C/N$ for $C = 1, 4, 16$. The monolithic multigrid preconditioner seems remarkably robust with respect to time step and mesh size.

	RadauIIA(1)		RadauIIA(2)		RadauIIA(3)	
Δt	time(s)	error	time(s)	error	time(s)	error
$\frac{1}{128}$	7.75	1.1×10^{-3}	15.86	2.8×10^{-4}	29.80	2.8×10^{-4}
$\frac{1}{32}$	2.72	4.1×10^{-3}	5.05	2.8×10^{-4}	8.56	2.8×10^{-4}
$\frac{1}{8}$	1.22	1.7×10^{-2}	2.01	2.9×10^{-4}	3.19	2.8×10^{-4}
$\frac{1}{2}$	0.8	6.7×10^{-2}	1.33	2.0×10^{-3}	2.02	3.3×10^{-4}

Table 2. Time-to-solution for the heat equation and L^2 error at the final time step using a range of different time steps for RadauIIA methods with 1, 2, and 3 stages. The multi-stage methods yield a lower error with less run-time than the 1-stage method.

methods. The reported time includes setting up the solver and preconditioner (a 1-time cost for linear problems) as well as solving the solution at each time step. We see that errors almost as small as they could be are obtained by the 2- and 3-stage methods with very large time steps in about 2-3 seconds of run-time, while the 1-stage method with small time steps takes several times as long to yield a worse solution.

4.3 The wave equation

We now turn to the wave equation (2.22). Here, our experiment fixes a 10×10 mesh of the unit square subdivided into right triangles with $RT_2 \times DG_1$ spatial discretization and considers the effect of the time-stepping method and step size on energy conservation. The semidiscrete system is known [Kirby and Kieu 2015] to exactly preserve the energy

$$E(t) = \frac{1}{2} \left(\|u\|^2 + \|\sigma\|^2 \right), \quad (4.1)$$

so that $E(t) = E(0)$ for all $t > 0$.

In particular, we pick $\Delta t = c/N$ for $c = 1, 5, 10$ so that $\Delta t = 0.1, 0.5, 1.0$. We consider the two lowest-order methods each of Gauss–Legendre, LobattoIIIC, and RadauIIA. Our experiment is simple: we integrate from $t = 0$ to $t = 10$ and compute the ratio of the final energy at $t = 10$ to the initial energy. These ratios are shown in Table 3. Although Gauss–Legendre methods fared poorly relative to L-stable methods for the heat equation, the reverse is true here. We see the exact energy conservation obtained for the Gauss–Legendre methods, while significant damping occurs for the LobattoIIIC and RadauIIA methods.

So far, all of our examples have used fully implicit methods, but here we point out how we can make use of PETSc options to achieve greater efficiency for DIRKs. Although this example is in the context of the wave equation, it applies equally to any setting in which a DIRK is appropriate.

method	Δt		
	0.1	0.5	1.0
Gauss–Legendre(1)	1	1	1
Gauss–Legendre(2)	1	1	1
LobattoIIIC(2)	3.79×10^{-1}	9.75×10^{-18}	1.17×10^{-20}
LobattoIIIC(3)	9.99×10^{-1}	5.19×10^{-2}	3.65×10^{-9}
RadauIIA(1)	1.50×10^{-8}	3.40×10^{-16}	6.79×10^{-14}
RadauIIA(2)	9.00×10^{-1}	7.10×10^{-4}	3.77×10^{-7}

Table 3. Energy conservation for the next-to-lowest order mixed method for the wave equation (2.22). We take a 10×10 mesh divided into right triangles and advance in time to $t = 10$ using various time-stepping strategies with various time steps. We see that the symplectic Gauss–Legendre methods with 1 and 2 stages conserve the energy to machine precision. However, the other methods do not conserve energy and become very dissipative as the time step is increased.

```

params = {"mat_type": "aij",
          "snes_type": "ksponly",
          "ksp_type": "preonly",
          "pc_type": "fieldsplit",
          "pc_fieldsplit_type": "multiplicative"}
params["pc_fieldsplit_0_fields"] = "0,1"
params["pc_fieldsplit_1_fields"] = "2,3"
per_field = {"ksp_type": "preonly",
             "pc_type": "lu"}
for i in range(butcher_tableau.num_stages):
    params["fieldsplit_%d" % i] = per_field

```

Listing 5. Simple DIRK-appropriate parameters for the mixed wave equation allowing one to solve for each stage in succession. A direct method is used on each stage.

We employ the Qin/Zhang 2-stage DIRK given by Butcher tableau

$$\begin{array}{c|cc}
 1/4 & 1/4 & 0 \\
 3/4 & 1/2 & 1/4 \\
 \hline
 & 1/2 & 1/2
 \end{array} \tag{4.2}$$

which is second-order, A-stable, and symplectic [Mei-Qing 1992]. Since the Butcher matrix A is lower-triangular, the system matrix (2.34) is block lower triangular, and so one may solve for each stage variable in succession via block forward substitution. This can be implemented as a PETSc preconditioner so that no special care is required in Irksome to handle DIRK methods differently. One obtains a block lower triangular preconditioner by means of a multiplicative field split [Brown et al. 2012], and if that preconditioner is applied exactly to a block lower triangular system, the system will be solved exactly.

Because Firedrake already has a mixed system (u and σ), we take care to specify that the system be blocked as a 2×2 system combining the variables for each stage rather than the default of a 4×4 split for each separate field. This can be controlled via PETSc options, and a solver specified for each block as in Listing 5. While our simple example applies a direct method on each diagonal block, more advanced solvers could also be employed. Repeating the experiments above for the wave equation with this Butcher tableau and these options gives identical energy behavior to the Gauss–Legendre methods in Table 3, although each time step simply consists of solving the diagonal blocks and performing forward substitution (all done inside the PETSc FIELDSPLIT preconditioner).

Although we have not undertaken a complete investigation, we believe that judicious use of PETSc options can obtain much of the efficiency that DIRKs offer. Using FIELDSPLIT makes the work comparable to traditional implementation — the stages are found successively. Also, we can bypass assembly of the off-diagonal blocks by using a matrix-free matrix format (`"mat_type": "matfree"`) and then using a `firedrake.AssembledPC` to assemble the diagonal blocks, if needed. This can also bring the memory footprint into line with a more traditional implementation.

While wave equations are also frequently integrated with explicit time-stepping methods, obtaining full efficiency with Irksome may be difficult. The main issue is related to finite element discretizations rather than Irksome per se. Unlike finite difference methods, explicit time integration with finite elements requires a solve with a mass matrix at each stage. One can use PETSc options to deploy a lightweight solver (for example, a diagonal field split with Jacobi-preconditioned conjugate gradients on the blocks). If a mass-lumping technique is available (for example, the Gauss–Lobatto technique is available in Firedrake [Homolya et al. 2017] and some lumped simplicial elements [Geevers et al. 2018] have been recently added), one step of Jacobi iteration (without a Krylov accelerator) will invert the mass matrices, but this will probably be less efficient than coding to a lower-level interface.

Finally, it should also be possible to develop stage-coupled preconditioners for fully implicit methods. A single-stage method for (2.22) leads to a linear system similar to a mixed form of the definite Helmholtz operator, so that would provide a starting point. A monolithic multigrid approach of Vanka-type as in [Molenaar 1991] seems more readily extensible to the multi-stage case than the weighted-norm preconditioners in [Arnold et al. 1997], but we leave such investigation to the future.

4.4 Nonlinear examples

We conclude our examples with three nonlinear problems. The time-dependent Navier–Stokes equations highlight a nonlinear DAE-type system. The Benjamin–Bona–Mahony dispersive wave model illustrates a nonlinear Sobolev-type equation. Finally, we use the Gross–Pitaevskii equation to investigate the applicability of the monolithic multigrid scheme proposed in Section 4.2 to a more complex three-dimensional problem of physical interest. Irksome’s high-level interface works unchanged in these examples.

4.4.1 Navier–Stokes. We consider the drag and lift calculations from the two-dimensional benchmark flow around a cylinder proposed in John [2004]; Schäfer et al. [1996]. The domain is given by $\Omega = [0, 2.2] \times [0, 0.41] \setminus B_r(0.2, 0.2)$, with radius $r = 0.05$, and is shown in Figure 4. The density is taken as $\rho = 1$ and kinematic viscosity is $\nu = 10^{-3}$. No-slip conditions are imposed on the top and bottom of the pipe and the cylinder. The natural boundary condition associated with the Laplacian formulation of the viscous term is imposed on the outflow boundary on the right end, and a parabolic profile is posed on the inflow boundary on the left end:

$$u(t, y) = \left(\frac{4U(t)y(0.41 - y)}{0.41^2}, 0 \right) \equiv \gamma(y, t), \quad (4.3)$$

where $U(t) = 1.5 \sin\left(\frac{\pi t}{8}\right)$. This configuration corresponds to a Reynolds number of 100. The problem is integrated over one time period, from $t = 0$ until $t = 8$.

We model this geometry using curvilinear triangles. Using Firedrake’s integration with OpenCascade, we are able to mesh the geometry with gmsh [Geuzaine and Remacle 2009] and refine the mesh in a geometrically conforming way. Our sample run uses a mesh with 14,640 cells and 7,546 vertices and a Scott–Vogelius discretization [Scott and Vogelius 1985] with P^4 velocities and discontinuous P^3 pressures. This amounts to about 382k global degrees of freedom. This higher order discretization also suggests a higher order time-discretization, and we use the two-stage RadauIIA

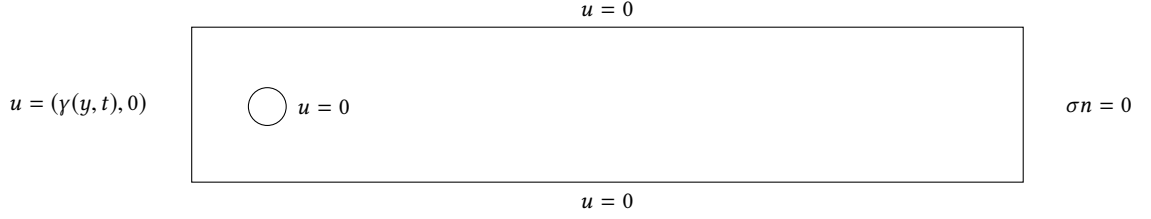


Fig. 4. Computational domain for flow past a cylinder, with boundary conditions indicated on each part of the boundary.

method with $\Delta t = 0.00125$. By contrast, the benchmark values to which we compare are obtained on a finer mesh (133,120 straight quadrilateral elements with 133,952 vertices) and lower-order discretization (biquadratic velocity and discontinuous linear pressure) with about 667k degrees of freedom. They use Crank-Nicolson time-stepping with $\Delta t = 1/1600 = 0.000625$, half the size of our time step. In Figure 5, we report the computed drag and lift for our method over time compared to the benchmark values, and excellent agreement is attained.

As a remark, Scott-Vogelius discretizations enforce the divergence-free condition pointwise in affine geometry, but this is not the case with our curvilinear mesh. While the exact theory of this method is still not understood, we have found it to be stable. The L^2 norm of the divergence at each time step was found to range between about 10^{-5} and 10^{-3} , which is comparable to the difference with the reference drag and lift values. If an exactly divergence-free method is required, one could reduce the geometry representation to affine, switch to an $H(\text{div})/L^2$ discretization [Cockburn et al. 2007], or employ the method recently proposed in Neilan and Otus [2020].

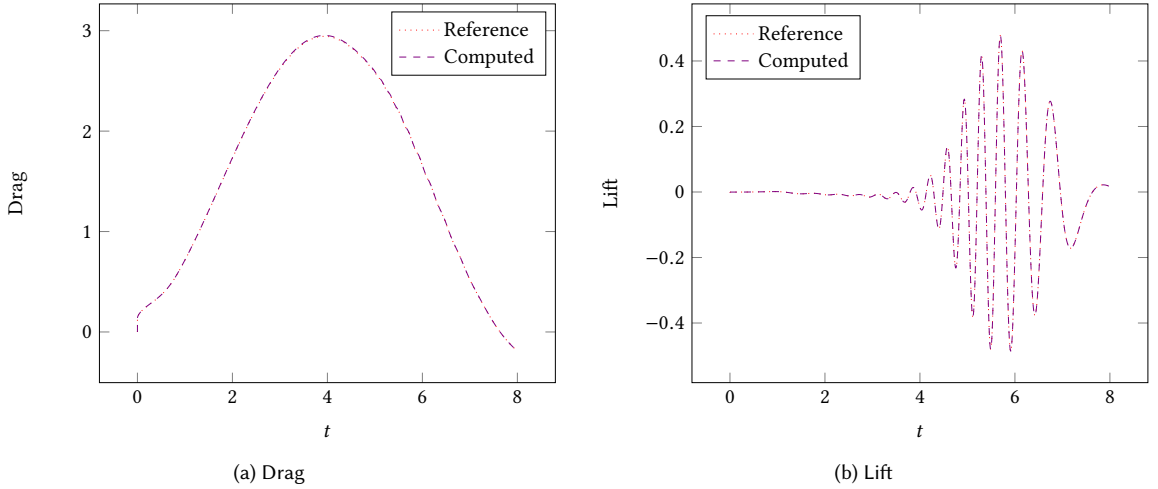


Fig. 5. Lift and drag calculations using Firedrake and Irksome compared to benchmark results in [John 2004; Schäfer et al. 1996]. Excellent agreement is obtained with half as many degrees of freedom and a time-step twice as large, owing to the use of higher-order methods.

4.4.2 Benjamin–Bona–Mahony. We consider the Benjamin–Bona–Mahony equation (2.25). With the Dt operator added in Irksome, we can express the weak form of the semidiscrete equation in UFL as

```
F = (inner(Dt(u), v) * dx + inner(u.dx(0), v) * dx
      + inner(u * u.dx(0), v) * dx + inner((Dt(u)).dx(0), v.dx(0)) * dx)
```

We consider this problem on the space interval $[0, 100]$ divided into $N = 1000$ intervals (giving mesh size $h = 0.1$) and pose periodic boundary conditions. We take the initial condition as

$$u(x, 0) = 3 \frac{c^2}{1 - c^2} \operatorname{sech}^2 \frac{1}{2} (cx + \delta),$$

where δ is chosen so that the bump is centered at $x = 40$ and $c = \frac{1}{2}$. This is the initial condition for a right-traveling solitary wave with velocity $\frac{1}{1 - c^2} = \frac{4}{3}$. We integrate from time 0 to time $T = 18$, at which the wave has traveled 24 units to the right.

We discretize the problem with a standard Galerkin method using P^1 finite elements and, given the Hamiltonian nature of the problem, Gauss–Legendre time stepping methods. With the implicit midpoint rule, we considered both $\Delta t = h$ and $\Delta t = 10h$. We found that the former case gave us about 0.15% relative L^2 error at $T = 18$, but the latter case with a large time step gave us greater than 10% relative error. Using the fourth-order, two-stage Gauss–Legendre method with $\Delta t = 10h$ actually gave us 0.14% relative error at the final time – less than the lower order method with a much smaller time step. Figure 6 shows the solution at the initial and final times as well as the pointwise error between the true and computed solutions at $t = 18$.

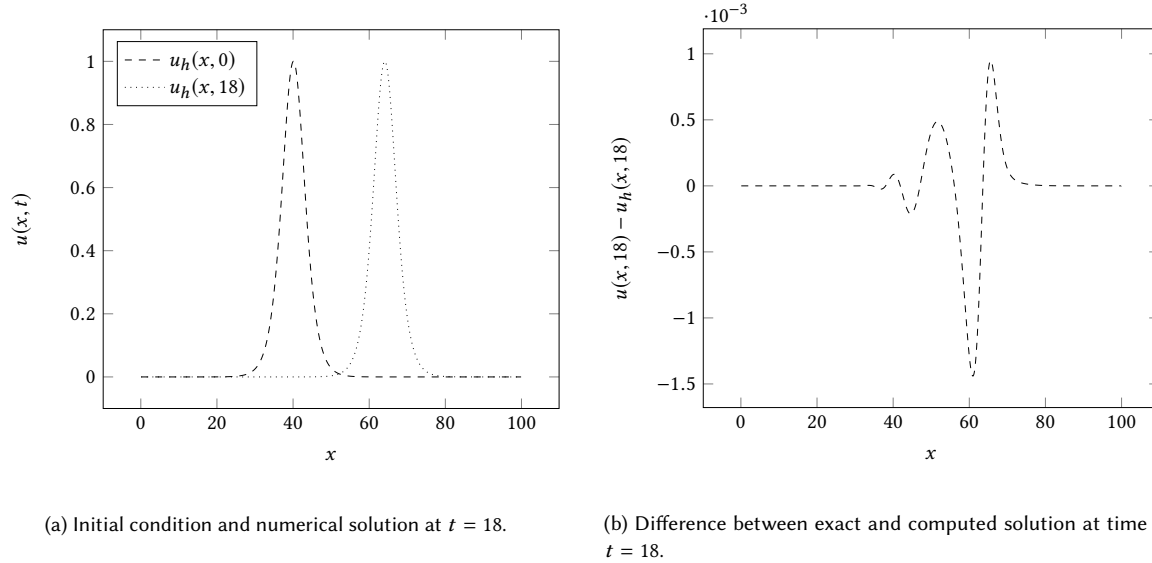


Fig. 6. Simulation of the Benjamin–Bona–Mahony problem using P^1 elements and GL(2) time-stepping methods.

We also demonstrate conservation of the invariants in Table 4. Both I_1 and I_2 were conserved to $O(10^{-15})$ for both GL methods. I_3 drifted slightly with both methods, on the order of 10^{-3} for the one-stage method (still, less than a 1% drift) and on the order of 10^{-6} for the two-stage method. While Gauss–Legendre methods are symplectic and preserve quadratic invariants [Hairer et al. 2006; Sanz-Serna 1988], very strong conditions are required for them to preserve cubic invariants as well [Iserles and Zanna 2000].

Method	t	$ 1 - I_1(t)/I_1(0) $	$ 1 - I_2(t)/I_2(0) $	$ 1 - I_3(t)/I_3(0) $
Gauss-Legendre(1)	0	0	0	0
	6	0	4.4×10^{-16}	1.5×10^{-3}
	12	2.2×10^{-16}	3.3×10^{-16}	3.7×10^{-3}
	18	2.2×10^{-16}	0	5.1×10^{-3}
Gauss-Legendre(2)	0	0	0	0
	6	2.2×10^{-16}	3.3×10^{-16}	4.3×10^{-7}
	12	0	1.1×10^{-16}	8.0×10^{-7}
	18	2.2×10^{-16}	4.4×10^{-16}	9.6×10^{-7}

Table 4. Conservation of invariants for the BBM equation (2.25) using one- and two-stage Gauss–Legendre methods. Invariant I_1 is linear and I_2 is quadratic, and these are exactly preserved, up to machine precision. The cubic invariant I_3 is not exactly conserved, but only very small relative deviation from the initial quantity is observed.

4.4.3 Gross–Pitaevskii. In recent work, Boullé et al. [2020] employed a bifurcation analysis technique to identify a large number of previously unknown stationary states of the Gross–Pitaevskii equation in a parabolic trap (2.29). These authors investigated the transient dynamics of a handful of unstable solutions by applying the energy-conserving integrator of Delfour et al. [1981] with initial condition given by a stationary state perturbed along its most unstable eigendirection. In this example we consider the same problem discretized with the Gauss–Legendre family of implicit RK methods, with the aim of investigating the performance of the monolithic multigrid scheme proposed in Section 4.2 for the arising block-dense linear systems. We consider a different base state to those described in Boullé et al. [2020].

For convenience, the Gross–Pitaevskii equation was written as a coupled PDE system for the real and imaginary components of the wavefunction ψ (although direct support for complex arithmetic has recently been included in Firedrake). Following Boullé et al. [2020], the domain $[-6, 6]^3$ was meshed with a uniform $10 \times 10 \times 10$ hexahedral grid, constructed by refining a coarse $5 \times 5 \times 5$ grid once. Both variables were approximated with continuous Lagrange elements of degree 3. For details of the construction of the initial condition from a base solution, see Boullé et al. [2020, Eq. (7)–(10)].

It is very desirable to conserve the invariants (2.30) in the time discretization. Since invariant J_1 is quadratic, symplectic integrators such as the Gauss–Legendre family will conserve it to machine precision (or at least algebraic solver tolerance). Although this is not the case for J_2 , we expect it to be well-conserved with higher-order methods. We vary the number of stages from $s = 1$ to $s = 3$ and the time step employed from $\Delta t = 0.05$ to $\Delta t = 0.25$. The time integration was conducted until $T = 30$ nondimensional time units.

As explained in Section 3, a key advantage of our symbolic manipulation is that it composes elegantly with the powerful solver functionality offered by Firedrake. We demonstrate this by employing the monolithic multigrid scheme proposed in Section 4.2 for the linear systems arising in the time integration of the Gross–Pitaevskii equation. The relaxation is implemented using PETSc’s PCASM. Full multigrid cycles are employed, with flexible GMRES used as the outermost Krylov solver Saad [1993]. Each outermost Krylov solve was terminated when the Euclidean norm of the residual fell below 10^{-8} or decreased by five orders of magnitude, whichever came first.

Images of the solution at times $t = 0, 10, 12, 14, 16, 18$ are given in Figure 7. The solution is visualized by rendering two isosurfaces of the wavefunction magnitude $|\psi|$, one for $|\psi| = 0.3$ and one for $|\psi| = 0.35$. The state at $t = 0$ exhibits two joined perpendicular vortex rings, with five vortex lines (cf. Boullé et al. [2020, Fig. 9b]). Four of the vortex lines are tangential to the two vortex rings, while one intersects them. The perturbation is not visible. For $t \in (0, 10)$, the

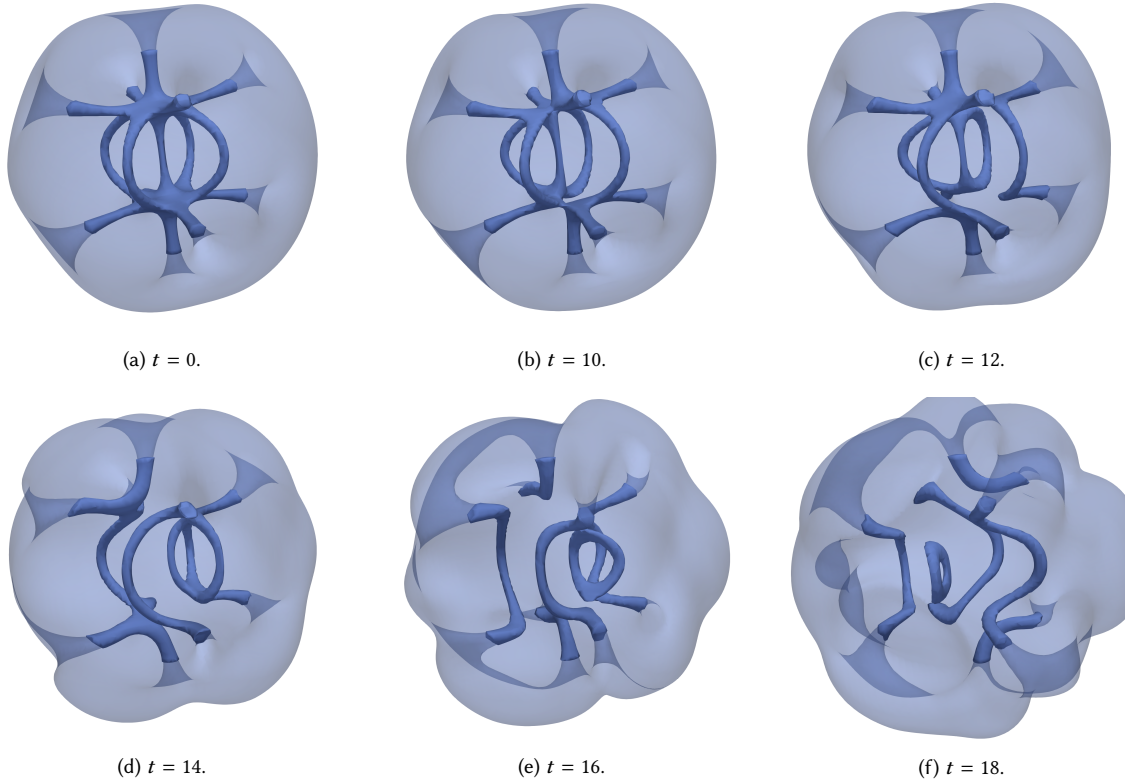


Fig. 7. Isosurfaces of the wavefunction magnitude $|\psi|$.

wavefunction oscillates in the planes spanned by the tangential vortex lines, with the magnitude of the oscillation increasing. At approximately $t \approx 10$, the vortex rings disconnect. At this point the dynamics become substantially more complex, with a large number of disconnections and reconnections, forming U-shaped vortex lines with multiple handles, vortex rings, and other structures. The last vortex rings terminate at around $t \approx 23$, and the solution thereafter consists of four vortex lines, oscillating strongly and connecting and disconnecting with one another.

Statistics about the conservation of J_2 and the solution procedure are given in Table 5. As expected, the higher-order methods are much better at preserving the quartic invariant J_2 . Moreover, the proposed multigrid scheme remains robust as both the time step Δt and number of stages s is varied; the number of Krylov iterations required per Newton step varies very little, from approximately 4.8 to 6.8 over the parameters considered. This further suggests that the theoretical investigation of monolithic multigrid schemes for implicit Runge–Kutta methods should be pursued. Similar results were achieved for unreported calculations with other base states, indicating that the proposed scheme is also robust to the specific initial condition used.

5 CONCLUSIONS AND FUTURE WORK

Irksome offers the opportunity for many future research directions. IMEX-type RK methods can be helpful with processes with multiple time scales (e.g. diffusion and advection) operating on the same variables. Some additional

GL(1)			GL(2)			GL(3)		
Δt	$ 1 - J_2(T)/J_2(0) $	# iters.	Δt	$ 1 - J_2(T)/J_2(0) $	# iters.	Δt	$ 1 - J_2(T)/J_2(0) $	# iters.
0.25	7.56×10^{-2}	4.88	0.25	2.09×10^{-2}	5.75	0.25	6.19×10^{-4}	6.58
0.10	2.25×10^{-2}	4.86	0.10	3.13×10^{-4}	5.42	0.10	3.31×10^{-6}	6.76
0.05	3.32×10^{-3}	5.40	0.05	2.00×10^{-5}	6.29	0.05	5.45×10^{-8}	6.79

Table 5. Conservation of energy (2.30) and Krylov iteration counts for the GP equation (2.29) using one, two, and three stage Gauss–Legendre methods. The last column in each table gives the number of Krylov iterations per Newton step, averaged over all time steps.

extensions to UFL, such as the form labeling present in Gusto [Ham et al. 2017], would be helpful in allowing users to segregate the terms to be treated implicitly and explicitly. More broadly, general linear methods [Butcher 2006] generalize both RK and multi-step methods. It is conceivable that general linear methods could be implemented by transforming UFL in the manner we have proposed here for RK methods. Lastly, it appears that monolithic multigrid schemes offer the potential for fast preconditioners for the linear systems arising in implicit RK methods, addressing one of the main obstacles to their wider use.

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/R029423/1 and EP/V001493/1]; and the National Science Foundation, award number 1912653.

REFERENCES

- Shrirang Abhyankar, Jed Brown, Emil Constantinescu, Debojyoti Ghosh, Barry Smith, and Hong Zhang. 2018. PETSc/TS: a modern scalable ODE/DAE solver library. (2018). arXiv:1806.01437.
- Roger Alexander. 1977. Diagonally implicit Runge–Kutta methods for stiff ODEs. *SIAM Journal on Numerical Analysis* 14, 6 (1977), 1006–1021.
- Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. 2014. Unified Form Language: a domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Software* 40, 2 (2014), 9:1–9:37. <https://doi.org/10.1145/2566630> arXiv:1211.4047
- Douglas N. Arnold, Richard S. Falk, and R. Winther. 1997. Preconditioning in $H(\text{div})$ and Applications. *Math. Comput.* 66, 219 (July 1997), 957–984. <https://doi.org/10.1090/S0025-5718-97-00826-0>
- Francesco Ballarin, Alberto Sartori, and Gianluigi Rozza. 2015. RBniCS-reduced order modelling in FEniCS. *ScienceOpen Posters* (2015).
- Thomas Brooke Benjamin, Jerry Lloyd Bona, and John Joseph Mahony. 1972. Model equations for long waves in nonlinear dispersive systems. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 272, 1220 (1972), 47–78.
- Theodore A. Bickart. 1977. An efficient solution process for implicit Runge–Kutta methods. *SIAM J. Numer. Anal.* 14, 6 (1977), 1022–1027. <https://doi.org/10.1137/0714069>
- Nicolas Boullé, Efstathios G. Charalampidis, Patrick E. Farrell, and Panayotis G. Kevrekidis. 2020. Deflation-based Identification of Nonlinear Excitations of the 3D Gross–Pitaevskii equation. *Physical Review A* 102 (2020), 053307. Issue 5. <https://doi.org/10.1103/PhysRevA.102.053307> arXiv:2004.10446.
- Jed Brown, Matthew G. Knepley, David A. May, Lois Curfman McInnes, and Barry Smith. 2012. Composable linear solvers for multiphysics. In *2012 11th International Symposium on Parallel and Distributed Computing*. IEEE, 55–62.
- John C. Butcher. 1976. On the implementation of implicit Runge–Kutta methods. *BIT Numerical Mathematics* 16, 3 (1976), 237–240.
- John C. Butcher. 2006. General linear methods. *Acta Numerica* 15 (2006), 157–256.
- Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. 2007. A note on discontinuous Galerkin divergence-free solutions of the Navier–Stokes equations. *Journal of Scientific Computing* 31, 1–2 (2007), 61–73.
- Germund G. Dahlquist. 1963. A special stability problem for linear multistep methods. *BIT Numerical Mathematics* 3, 1 (1963), 27–43.
- Andreas Dedner, Robert Klöforn, Martin Nolte, and Mario Ohlberger. 2010. A generic interface for parallel and adaptive discretization schemes: abstraction principles and the DUNE-FEM module. *Computing* 90, 3–4 (2010), 165–196.
- Michel Delfour, Michel Fortin, and Guy Payre. 1981. Finite-difference solutions of a non-linear Schrödinger equation. *J. Comput. Phys.* 44, 2 (1981), 277–288. [https://doi.org/10.1016/0021-9991\(81\)90052-8](https://doi.org/10.1016/0021-9991(81)90052-8)

- Patrick E. Farrell, David A. Ham, Simon W. Funke, and Marie E. Rognes. 2013. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing* 35, 4 (2013), C369–C393.
- Patrick E. Farrell, Matt G. Knepley, Lawrence Mitchell, and Florian Wechsung. 2020. PCPATCH: Software for the topological construction of multigrid relaxation methods. *ACM Trans. Math. Software* (2020). Accepted. arXiv:1912.08516.
- Sjoerd Geevers, Wim A Mulder, and Jaap JW van der Vegt. 2018. New higher-order mass-lumped tetrahedral elements for wave propagation modelling. *SIAM journal on scientific computing* 40, 5 (2018), A2830–A2857. <https://doi.org/10.1137/18M1175549>
- Christophe Geuzaine and Jean-François Remacle. 2009. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331. <https://doi.org/10.1002/nme.2579>
- Tunc Geveci. 1988. On the application of mixed finite element methods to the wave equation. *Math. Model. Numer. Anal* 22 (1988), 243–250.
- Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. 2001. Strong stability-preserving high-order time discretization methods. *SIAM Rev.* 43, 1 (2001), 89–112.
- Eugene P. Gross. 1961. Structure of a quantized vortex in boson systems. *Il Nuovo Cimento* 20 (1961), 454–477. Issue 3. <https://doi.org/10.1007/BF02731494>
- Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Vol. 31. Springer Science & Business Media.
- David Ham, Lawrence Mitchell, Miklós Homolya, Fabio Luporini, Thomas Gibson, Paul Kelly, Colin Cotter, Michael Lange, Stephan Kramer, Jemma Shipton, Hiroe Yamazaki, Alberto Paganini, and Tuomas Kärnä. 2017. Automating the generation of finite element dynamical cores with Firedrake. In *EGU General Assembly Conference Abstracts*, Vol. 19. 17987.
- Jan S. Hesthaven, Gianluigi Rozza, and Benjamin Stamm. 2016. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. Vol. 590. Springer.
- Miklós Homolya, Robert C. Kirby, and David A. Ham. 2017. Exposing and exploiting structure: optimal code generation for high-order finite element methods. (2017). arXiv:1711.02473
- Weizhang Huang, Lennard Kamenski, and Jens Lang. 2019. Conditioning of implicit Runge–Kutta integration for finite element approximation of linear diffusion equations on anisotropic meshes. *J. Comput. Appl. Math.* (2019), 112497.
- Arieh Iserles and Antonella Zanna. 2000. Preserving algebraic invariants with Runge–Kutta methods. *J. Comput. Appl. Math.* 125, 1-2 (2000), 69–81.
- Volker John. 2004. Reference values for drag and lift of a two-dimensional time-dependent flow around a cylinder. *International Journal for Numerical Methods in Fluids* 44, 7 (2004), 777–788.
- Tate Kernell and Robert C Kirby. 2020. Preconditioning mixed finite elements for tide models. (2020). arXiv:2003.01632.
- Panayotis G. Kevrekidis, Dimitri J. Frantzeskakis, and Ricardo Carretero-González. 2015. *The Defocusing Nonlinear Schrödinger Equation*. SIAM. <https://doi.org/10.1137/1.9781611973945>
- Robert C. Kirby. 2004. Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Software* 30, 4 (2004), 502–516. <https://doi.org/10.1145/1039813.1039820>
- Robert C. Kirby and Thanh Tri Kieu. 2015. Symplectic-mixed finite element approximation of linear acoustic wave equations. *Numer. Math.* 130, 2 (2015), 257–291.
- Robert C. Kirby and Lawrence Mitchell. 2018. Solver composition across the PDE/linear algebra barrier. *SIAM Journal on Scientific Computing* 40, 1 (2018), C76–C98. <https://doi.org/10.1137/17M1133208>
- Butler W. Lampson and Howard E. Sturgis. 1976. Reflections on an operating system design. *Commun. ACM* 19, 5 (1976), 251–265.
- Michael Lange, Lawrence Mitchell, Matthew G Knepley, and Gerard J Gorman. 2016. Efficient mesh management in firedrake using PETSC DMPLEX. *SIAM Journal on Scientific Computing* 38, 5 (2016), S143–S155.
- Anders Logg. 2003. Multi-adaptive Galerkin methods for ODEs I. *SIAM Journal on Scientific Computing* 24, 6 (2003), 1879–1902.
- Anders Logg, Kent-Andre Mardal, and Garth N. Wells (Eds.). 2012. *Automated solution of differential equations by the finite element method: the FEniCS book*. Vol. 84. Springer. <https://doi.org/10.1007/978-3-642-23099-8>
- James R. Maddison and Patrick E. Farrell. 2014. Rapid development and adjoining of transient finite element models. *Computer Methods in Applied Mechanics and Engineering* 276 (2014), 95–121.
- Kent-Andre Mardal, Trygve K. Nilssen, and Gunnar Andreas Staff. 2007. Order-optimal preconditioners for implicit Runge–Kutta schemes applied to parabolic PDEs. *SIAM Journal on Scientific Computing* 29, 1 (2007), 361–375.
- Zhang Mei-Qing. 1992. Diagonally Implicit Symplectic Runge–Kutta schemes for Hamiltonian systems.. In *Scientific Computation-Proceedings Of International Conference*, Vol. 1. World Scientific, 259.
- Lawrence Mitchell and Eike Hermann Müller. 2016. High level implementation of geometric multigrid solvers for finite element problems: Applications in atmospheric modelling. *J. Comput. Phys.* 327 (2016), 1–18.
- Johannes Molenaar. 1991. A two-grid analysis of the combination of mixed finite elements and Vanka-type relaxation. In *Multigrid Methods III*. Springer, 313–323.
- Michael Neilan and M. Baris Otus. 2020. Divergence-free Scott–Vogelius elements on curved domains. (2020). arXiv:2008.06429
- Luca F. Pavarino. 1993. Additive Schwarz methods for the p -version finite element method. *Numer. Math.* 66, 1 (1993), 493–515. <https://doi.org/10.1007/BF01385709>
- Will Pazner and Per-Olof Persson. 2017. Stage-parallel fully implicit Runge–Kutta solvers for discontinuous Galerkin fluid simulations. *J. Comput. Phys.* 335 (2017), 700–717.
- Lev P. Pitaevskii. 1961. Vortex lines in an imperfect Bose gas. *Soviet Physics JETP* 13, 2 (1961), 451–454.

- Md Masud Rana, Victoria E. Howle, Katharine Long, Ashley Meek, and William Milestone. 2020. A New Block Preconditioner for Implicit Runge-Kutta Methods for Parabolic PDE. *arXiv preprint arXiv:2010.11377* (2020).
- Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. 2016. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Software* 43, 3 (2016), 24:1–24:27. <https://doi.org/10.1145/2998441> arXiv:1501.01809
- Yousef Saad. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing* 14, 2 (1993), 461–469. <https://doi.org/10.1137/0914028>
- Jesus M. Sanz-Serna. 1988. Runge–Kutta schemes for Hamiltonian systems. *BIT Numerical Mathematics* 28, 4 (1988), 877–883.
- Michael Schäfer, Stefan Turek, Franz Durst, Egon Krause, and Rolf Rannacher. 1996. Benchmark computations of laminar flow around a cylinder. In *Flow simulation with high-performance computers II*. Springer, 547–566.
- Joachim Schöberl, Jens M. Melenk, Clemens Pechstein, and Sabine Zaglmayr. 2008. Additive Schwarz preconditioning for p -version triangular and tetrahedral finite elements. *IMA J. Numer. Anal.* 28 (2008), 1–24. <https://doi.org/10.1093/imanum/drl046>
- L. Ridgway Scott and Michael Vogelius. 1985. Norm estimates for a maximal right inverse of the divergence operator in spaces of piecewise polynomials. *ESAIM: Mathematical Modelling and Numerical Analysis* 19, 1 (1985), 111–143.
- Gideon Simpson and Marc Spiegelman. 2011. Solitary wave benchmarks in magma dynamics. *Journal of Scientific Computing* 49, 3 (2011), 268–290.
- Gunnar A. Staff, Kent-Andre Mardal, and Trygve K. Nilssen. 2006. Preconditioning of fully implicit Runge-Kutta schemes for parabolic PDEs. *Modeling, Identification, and Control* 27, 1 (2006), 109–123.
- Tianjiao Sun, Lawrence Mitchell, Kaushik Kulkarni, Andreas Klöckner, David A Ham, and Paul HJ Kelly. 2020. A study of vectorization for matrix-free finite element methods. *The International Journal of High Performance Computing Applications* 34, 6 (2020), 629–644.
- J. Van Lent and S. Vandewalle. 2005. Multigrid Methods for Implicit Runge–Kutta and Boundary Value Method Discretizations of Parabolic PDEs. *SIAM Journal on Scientific Computing* 27, 1 (2005), 67–92. <https://doi.org/10.1137/030601144>
- Umberto Villa, Noemi Petra, and Omar Ghattas. 2018. hiPPYlib: An extensible software framework for large-scale inverse problems. *Journal of Open Source Software* 3, 30 (2018), 940.
- Gerhard Wanner and Ernst Hairer. 1996. *Solving ordinary differential equations II*. Springer Berlin Heidelberg.