Fluid Dynamics and Co-located Conferences
June 24-27, 2013, San Diego, CA
21st AIAA Computational Fluid Dynamics Conference

Performance tuning of Newton-GMRES methods for discontinuous Galerkin discretizations of the Navier-Stokes equations

Matthew Zahr*
Stanford University, Stanford, CA 94305, U.S.A.
Per-Olof Persson†
University of California, Berkeley, Berkeley, CA 94720-3840, U.S.A.

In this work, we investigate numerical solvers and time integrators for the system of Ordinary Differential Equations (ODEs) arising from the Discontinuous Galerkin Finite Element Method (DG-FEM) semi-discretization of the Navier-Stokes equations to explore potential speedup opportunities. DG-FEMs have many desirable properties such as stability, high-order accuracy, and ability to handle complex geometries. However, they are notorious for their high computational cost and storage. In this document, our problems are spatially discretized with DG-FEM, temporally discretized with various implicit ODE solvers, and the nonlinear systems are solved using a Newton-GMRES method [1, 2]. We study the effects of varying several parameters, including the ODE solver, predictors for Newton's method, GMRES tolerance, and Jacobian recycling. We show that by properly choosing these parameters, speedup factors of between 5 and 14 can be achieved over non-optimal choices. The numerical experiments are performed on two model flow problems: An Euler vortex and the viscous flow over a 2D NACA wing at a high angle of attack.

I. Introduction

High-order Discontinuous Galerkin Finite Element Methods (DG-FEM) for conservation laws have recently gained much attention due to their combination of several desirable features. They have the capability of achieving arbitrarily high orders of accuracy on complex, irregular domains and are provably stable for conservation laws. Other commonly used numerical solution methods for PDEs lack at least one of these properties: the Finite Element Method (FEM) is able to achieve arbitrarily high-orders of accuracy on general domains, but lack stability for conservation laws; the Finite Volume Method (FVM) is stable for conservation laws and can handle arbitrary domains, but generalization to high-order accuracy is not straightforward; and the Finite Difference Method (FDM) can achieve high-order accuracy and stability for conservation laws, but cannot handle arbitrary domains [3].

The many desirable features of DG-FEM come at the price of substantially higher storage demands and computational cost than some of the other methods. The additional storage can be traced to two obvious sources: (1) high-order accuracy and (2) redundant degrees of freedom at shared faces. The first source of additional storage is not unique to DG-FEM; it arises from the fact that high-order accurate simulations require elements with a large number of connected nodes for FEM and DG-FEM. The second storage source is a result of not explicitly enforcing continuity between adjacent elements and is unique to DG-FEM; this is also responsible for the stabilizing property of DG-FEM [4].

Discontinuous Galerkin methods are generally regarded as roughly an order of magnitude too computationally expensive, in terms of both storage and CPU time, for widespread practical use. In this work, the computational cost of DG-FEM, specifically CPU time, is addressed by investigating existing methods for

*Graduate Student, Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94035. E-mail: mzahr@stanford.edu. AIAA Student Member.
†Assistant Professor, Department of Mathematics, University of California, Berkeley, Berkeley CA 94720-3840. E-mail: persson@berkeley.edu. AIAA Member.

nontrivial speedup factors hidden in the details of the algorithms. The governing equations of the system are spatially discretized by DG-FEM and various temporal integrators are applied to solve the resulting system of Ordinary Differential Equations (ODE). Since the applications of interest for this work are viscous flow simulations, we only consider implicit time integration schemes. The application of an implicit ODE solver yields a sequence of nonlinear systems of equations, which are solved using a Newton-GMRES approach. At each step of the outlined procedure, there are many algorithmic choices that must be made that have a significant impact on the performance of the method. Among these algorithmic choices are: implicit ODE solver, predictors for Newton's method, Jacobian re-computation or recycling, GMRES tolerance, and GMRES preconditioner, to list a few.

In this work, the interplay between accuracy and CPU time for standard classes of implicit ODE solvers, including Backward Differentiation Formulas (BDF) [5, 6] and Diagonally-Implicit Runge-Kutta (DIRK)[7] schemes, is investigated. It is well-known that an ODE solver is chosen based on a trade-off between desired temporal accuracy and total computation time. This trade-off is quantified for two model problems (Euler vortex and viscous flow over NACA wing at high angle of attack) described in Section II. Furthermore, the effect of Newton-GMRES parameters such as GMRES tolerance, Newton predicting, and Jacobian recycling on CPU time is studied. The effect of these parameters on computational time is rarely examined in the literature, but we will show that their selection can result in dramatic differences in CPU time.

Many of the results of this work are not specific to DG-FEM since the starting point of our testing is the semi-discrete form of the governing equations. However, due to the higher computational expense per iteration of DG-FEM over FEM, FVM, and FDM, they have the most to gain from these hidden speedups.

The remainder of this paper is organized as follows. In Section II, we present the governing equations of interest, the Euler and Navier-Stokes equations, as well as the model problems we use to perform our investigation of the aforementioned methods. Section III is devoted to presenting the necessary formulation of the time-integration schemes considered. We also present the Newton-GMRES algorithm for solving a nonlinear system of equations and identify the parameters that will be the focus of our study. The heart of this work is contained in Section IV, where we present the numerical study of the effect of the identified parameters on CPU time for the two model problems. Finally, in Section V, we summarize our results and provide some general guidelines.

II. Governing Equations

Consider the compressible Euler and Navier-Stokes equations:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_i}(\rho u_i u_j + p) = +\frac{\partial \tau_{ij}}{\partial x_j} \quad \text{for } i = 1, 2, 3, \tag{2}$$

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_i}\left(u_j(\rho E + p)\right) = -\frac{\partial q_j}{\partial x_j} + \frac{\partial}{\partial x_j}(u_j \tau_{ij}), \tag{3}$$

where ρ is the fluid density, u_1, u_2, u_3 are the velocity components, and E is the total energy. The viscous stress tensor and heat flux are given by

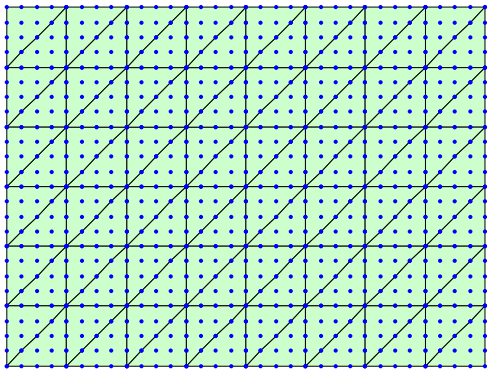
$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_j}\delta_{ij}\right) \qquad \text{and} \qquad q_j = -\frac{\mu}{\mathrm{Pr}}\frac{\partial}{\partial x_j}\left(E + \frac{p}{\rho} - \frac{1}{2}u_k u_k\right). \tag{4}$$

Here, μ is the viscosity coefficient and $\mathrm{Pr} = 0.72$ is the Prandtl number which we assume to be constant. For an ideal gas, the pressure p has the form

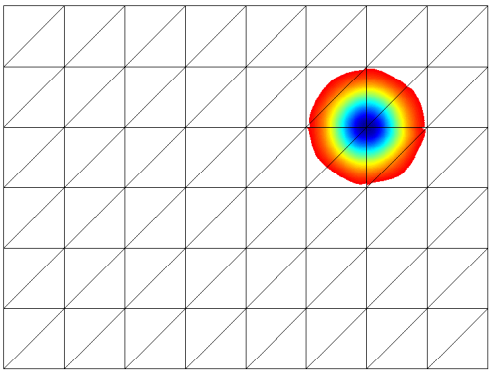
$$p = (\gamma - 1)\rho\left(E - \frac{1}{2}u_k u_k\right), \tag{5}$$

where γ is the adiabatic gas constant.

The first model problem is the inviscid flow of a compressible vortex in a rectangular domain. The vortex is initially centered at (x_0, y_0) and is moving with the free-stream at an angle θ with respect to the x-axis.

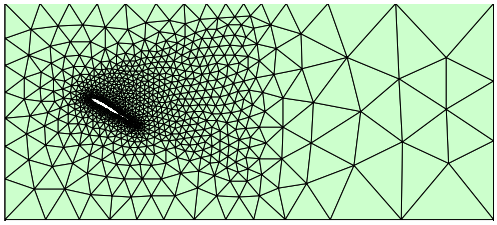


Euler vortex mesh, with degree $p = 4$

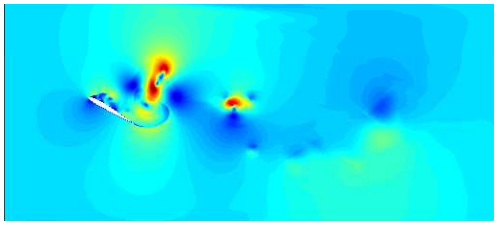


Solution (density)

Figure 1. Euler Vortex: Mesh and Solution at $t_0 = \sqrt{10^2 + 5^2}$



NACA mesh, with degree $p = 4$



Solution (Mach)

Figure 2. NACA Wing: Mesh and Solution at $t_0 = 5.01$

The analytic solution at (x, y, t) is given by

$$u = u_\infty \left(\cos\theta - \frac{\epsilon((y - y_0) - \bar{v}t)}{2\pi r_c} \exp(f/2) \right), \qquad \rho = \rho_\infty \left(1 - \frac{\epsilon^2(\gamma - 1)M_\infty^2}{8\pi^2} \exp(f) \right)^{\frac{1}{\gamma - 1}}, \tag{6}$$

$$v = u_\infty \left(\sin\theta + \frac{\epsilon((x - x_0) - \bar{u}t)}{2\pi r_c} \exp(f/2) \right), \qquad p = p_\infty \left(1 - \frac{\epsilon^2(\gamma - 1)M_\infty^2}{8\pi^2} \exp(f) \right)^{\frac{\gamma}{\gamma - 1}}, \tag{7}$$

where $f(x, y, t) = (1 - ((x - x_0) - \bar{u}t)^2 - ((y - y_0) - \bar{v}t)^2)/r_c^2$, M_∞ is the Mach number, $\gamma = c_p/c_v = 1.4$, and u_∞, p_∞, ρ_∞ are free-stream velocity, pressure, and density. The Cartesian components of the free-stream velocity are $\bar{u} = u_\infty \cos\theta$ and $\bar{v} = u_\infty \sin\theta$. The parameter ϵ measures the strength of the vortex and r_c is its size.

The computational domain is of size 20-by-15, with the vortex initially centered at $(x_0, y_0) = (5, 5)$ with respect to the lower-left corner. The Mach number is $M_\infty = 0.5$, the angle $\theta = \arctan(1/2)$, and the vortex has the parameters $\epsilon = 0.3$ and $r_c = 1.5$. We use characteristic boundary conditions and integrate until time $t_0 = \sqrt{10^2 + 5^2}$.

The second model problem is the viscous flow around a NACA wing at a high angle of attack. The initial condition for the simulation is free-stream flow throughout the domain. As the flow develops around the wing, there is substantial flow separation along the span of the wing due to the high angle of attack.

The fluid domain is of size 9-by-4 with the wing centered vertically in the domain, near the inlet boundary. This placement of the wing allows vortices to fully develop in the wake while remaining within the fluid domain. There is a no-slip condition on the wing surface and far-field conditions on all other boundaries; the Mach number and gas constant for this simulation are 0.2 and 1.4, respectively. We integrate until time $t = 5$, when the flow separation has caused vortices to develop in the wake of the wing. For our numerical experiments, the solution at $t = 5$ will be used as our initial condition and we integrate until time $t_0 = 5.01$ with the various schemes used in the study. Starting the simulation from $t = 5$ ensures that our numerical experiments are performed in an interesting regime of the flow and using only short-time integration (from $t = 5$ to $t_0 = 5.01$) avoids convergence issues associated with the chaotic behavior of the problem.

For both model problems, the spatial discretization is performed using DG-FEM with fourth-order elements. The Euler vortex mesh has 96 elements with 15 nodes per element, which corresponds to 1,440 nodes

and 5,760 degrees of freedom. The Euler vortex mesh and solution at $t_0 = \sqrt{10^2 + 5^2}$ are shown in Figure 1. The NACA wing mesh has 2,512 triangles with 15 nodes per element, which corresponds to 37,680 nodes and 150,720 degrees of freedom. The mesh of the NACA wing and the solution at $t_0 = 5.01$ are shown in Figure 2, respectively. All simulations presented in this work were performed using the 3DG software package [8,9], a fully parallel DG-FEM code, based on Compact Discontinuous Galerkin (CDG) [4] numerical fluxes, with MATLAB and Python interfaces, written by P.-O. Persson and J. Peraire.

As mentioned in Section I, the methods investigated in this paper operate on the semi-discrete or fully discrete version of the Navier-Stokes equations. Therefore, it is beneficial to pose the semi-discrete version of the (1) - (5) in an abstract sense. The spatial discretization of (1) - (5) can be written in the following form:

$$\mathbb{M}\dot{\mathbf{u}}(t) = \mathbf{r}(t, \mathbf{u}(t)), \tag{8}$$

where $\mathbb{M} \in \mathbb{R}^{N\times N}$ is the block diagonal mass matrix, each block corresponding to the mass contribution of an element in the domain, $\mathbf{r} : \mathbb{R}_+ \times \mathbb{R}^N \to \mathbb{R}^N$ is the discretization of the nonlinear terms in (1) - (5), and $\mathbf{u}(t) \in \mathbb{R}^N$ is the discrete state vector. Here, N is the number of degrees of freedom in the domain. We emphasize that N is typically very large for the reasons mentioned in Section I: high order spatial discretization ($p = 4$) and repeated degrees of freedom at nodes situated on shared faces.

III. Nonlinear Ordinary Differential Equation Solvers

III.A. Time Integration Schemes

In this section, we present the *implicit* time integration schemes considered in this work; all of these schemes will be applied to the semi-discrete equation (8). Application of an implicit time-solver to the nonlinear system of ODE in (8) results in a sequence on nonlinear systems of equations of the form:

$$\mathcal{R}(\mathbf{u}^{(n+1)}; t_{n+1}, \mathbf{u}^{(n)}, \ldots, \mathbf{u}^{(0)}) = 0 \tag{9}$$

where $\mathcal{R}$ is a vector-valued nonlinear function dependent on the integration scheme, Δt is the time step size (assumed constant for all time steps for notational convenience), and $\mathbf{u}^{(i)} = \mathbf{u}(t_i)$. This is the residual of the nonlinear system arising at step n from the implicit time discretization. Multi-stage schemes, such as those arising from Diagonally-Implicit Runge-Kutta methods, fall into the notation of (9) by defining $\mathcal{R}$ as the equation from each stage stacked on each other. For notational convenience, we will write the above residual as $\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)})$ with the dependence on time and previous time steps implied.

III.A.1. Backward Differentiation Formula (BDF) Methods

The BDF [5, 6] integration schemes are implicit, multi-step methods that combine previous time steps to achieve higher orders of accuracy. The main disadvantages of higher order BDF schemes is that initialization of the first several time steps must be done with an integration scheme that does not require solution history (either low-order BDF schemes or Runge-Kutta type schemes), making rigorous convergence studies somewhat difficult. Also, the highest order A-stable BDF scheme is BDF2, which is only second-order. In the DG-FEM community where high-order spatial accuracy is the goal, high-order time integration schemes are required so as not to limit the speed of convergence to the exact solution of the PDE. In this section, the standard BDF1 and BDF2 schemes are introduced, along with BDF23 (second-order, A-stable) and BDF23_3 (third-order, A-stable), new BDF-type schemes developed by K. Miller [10].

The general form of the BDF class applied to (8) is:

$$\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(\sum_{i=0}^{n} \alpha_i \mathbb{M}\mathbf{u}^{(i)} + \kappa\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)}; \mathbf{u}^{(n)}, \ldots, \mathbf{u}^{(0)})\right) = 0, \tag{10}$$

where $\alpha_0, \ldots, \alpha_n, \kappa \in \mathbb{R}$ are the coefficients that define the particular BDF scheme. The first scheme, BDF1, also known as Backward Euler is a single-step, A-stable, first-order accurate, implicit scheme. Application of BDF1 to (8) yields the following sequence of nonlinear equations:

$$\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(\mathbb{M}\mathbf{u}^{(n)} + \Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)})\right) = 0, \tag{11}$$

which fits in the framework defined by (10) with $\alpha_n = 1$, $\kappa = 1$, and $\alpha_0 = \cdots = \alpha_{n-1} = 0$. The standard second-order BDF scheme is a multi-step, A-stable scheme, known as BDF2:

$$\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(\frac{4}{3}\mathbb{M}\mathbf{u}^{(n)} - \frac{1}{3}\mathbb{M}\mathbf{u}^{(n-1)} + \frac{2}{3}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)}; \mathbf{u}^{(n)})\right) = 0. \tag{12}$$

BDF2 fits into the BDF structure with $\alpha_{n-1} = -\frac{1}{3}, \alpha_n = \frac{4}{3}, \kappa = \frac{2}{3}$, and $\alpha_0 = \cdots = \alpha_{n-2} = 0$. The third order BDF scheme, known as BDF3, falls into the BDF framework with $\alpha_{n-2} = \frac{2}{11}, \alpha_{n-1} = -\frac{9}{11}, \alpha_n = \frac{18}{11}, \kappa = \frac{6}{11}$, and $\alpha_0 = \cdots = \alpha_{n-3} = 0$ with the caveat that this scheme lacks A-stability, which limits its use in practice. For this reason, BDF3 is not included in any of the studies in this document. It was introduced purely as motivation for the new BDF-like schemes, BDF23 and BDF23_3. BDF23 is a relaxed, A-stable version of BDF3 that is second-order accurate and falls into the BDF framework with $\alpha_{n-2} = \tau b_3 + (1-\tau)a_3, \alpha_{n-1} = \tau b_2 + (1-\tau)a_2, \alpha_n = \tau b_1 + (1-\tau)a_1, \kappa = \tau\beta_2 + (1-\tau)\beta_3$, and $\alpha_0 = \cdots = \alpha_{n-3} = 0$, where b_1, b_2, b_3, and β_2 are the BDF2 coefficients

$$b_1 = \frac{4}{3}, b_2 = -\frac{1}{3}, b_3 = 0, \beta_2 = \frac{2}{3}, c_2 = -\frac{2}{9}$$

and a_1, a_2, a_3, and β_3 are the BDF3 coefficients

$$a_1 = \frac{18}{11}, a_2 = -\frac{9}{11}, a_3 = \frac{2}{11}, \beta_3 = \frac{6}{11}, c_3 = -\frac{3}{22}.$$

Define

$$\begin{aligned} bb_1 &= \tau b_1 + (1-\tau)a_1 \\ bb_2 &= \tau b_2 + (1-\tau)a_2 \\ bb_3 &= \tau b_3 + (1-\tau)a_3 \\ \beta_{23} &= \tau\beta_2 + (1-\tau)\beta_3 \\ c_{23} &= \tau c_2, \end{aligned}$$

then BDF23 can be written explicitly as

$$\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(bb_1\mathbb{M}\mathbf{u}^{(n)} + bb_2\mathbb{M}\mathbf{u}^{(n-1)} + bb_3\mathbb{M}\mathbf{u}^{(n-2)} + \beta_{23}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)})\right) = 0. \tag{13}$$

BDF23, first introduced by K. Miller, is an A-stable (if $\tau > 0.45$) linear combination between BDF2 and BDF3, where τ is the relaxation parameter.

Lastly, an A-stable, third-order BDF-like scheme introduced by K. Miller, known as BDF23_3, combines a predictor step of BDF23, corrected with BDF3. This is a two-stage scheme and therefore does not fit directly into the BDF framework in (10). The algorithm proceeds as follows

- Define $\mathbf{u}_{23}$ as
$$\mathbf{u}_{23} = bb_1\mathbf{u}^{(n)} + bb_2\mathbf{u}^{(n-1)} + bb_3\mathbf{u}^{(n-2)}$$
- Solve the nonlinear Backward Cauchy-Euler equation $\mathcal{R}(\mathbf{u}_i) = 0$ (i.e. using Newton-GMRES, Algorithm 1), where
$$\mathcal{R}(\mathbf{u}_i) = \mathbb{M}\mathbf{u}_i - (\mathbb{M}\mathbf{u}_{23} + \beta_{23}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}_i))$$
- Define $\mathbf{u}_{33}$ as
$$\mathbf{u}_{33} = a_1\mathbf{u}^{(n)} + a_2\mathbf{u}^{(n-1)} + a_3\mathbf{u}^{(n-2)} - \delta(\mathbf{u}_i - \mathbf{u}_{23})$$
where
$$\begin{aligned} \alpha &= \frac{c_3}{c_{23}} + \epsilon \\ \delta &= \frac{\alpha}{\beta_{23}} \end{aligned}$$

- Solve the nonlinear Backward Cauchy-Euler equation $\mathcal{R}(\mathbf{u}_{n+1}) = 0$ (i.e. using Newton-GMRES, Algorithm 1), where

$$\mathcal{R}(\mathbf{u}_{n+1}) = \mathbb{M}\mathbf{u}^{(n+1)} - \left(\mathbb{M}\mathbf{u}_{33} + \beta_{33}\Delta t\mathbf{r}(t_{n+1}, \mathbf{u}^{(n+1)})\right)$$

and

$$\beta_{33} = \beta_3 + \alpha$$

For the numerical experiments in Section IV, the parameters $\epsilon = 0.00063$ and $\tau = \frac{-b+\sqrt{b^2-4ac}}{2a}$ where

$$a = \beta_2 - \beta_3$$
$$b = -\epsilon$$
$$c = -\frac{c_3}{c_2}$$

are used, which uniquely define the BDF23_3 scheme. As far as we know, this work is the first to introduce the BDF23 and the BDF23_3 schemes to the DG-FEM and the CFD communities.

III.A.2. Diagonally-Implicit Runge-Kutta (DIRK) Methods

The DIRK [7] class of time-integration schemes is a special case of the implicit Runge-Kutta class where the Butcher Tableau is lower triangular. The DIRK schemes are single-step methods and take the following form

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \sum_{i=1}^{s} b_i \mathbf{k}_i$$

$$\mathbb{M}\mathbf{k}_i = \Delta t\mathbf{r}\left(t_n + c_i\Delta t, \mathbf{u}^{(n)} + \sum_{j=1}^{i} a_{ij}\mathbf{k}_j\right), \tag{14}$$

where s is the number of stages in the scheme. In the following, we will consider 3 specific DIRK schemes, that are A- and L-stable with the order of accuracy equal to the number of stages s. We denote the k-th order accurate DIRK scheme as DIRKk, i.e 1st order DIRK is DIRK1, etc. The Butcher tableaus for the DIRK schemes studied in this document are presented in Tables 1-3.

1	1
	1

Table 1. DIRK1 Butcher Tableau

α	α	0
1	1-α	α
	1-α	α

Table 2. DIRK2 Butcher Tableau
$\alpha = 1 - \frac{1}{\sqrt{2}}$

α	α		
$\frac{1+\alpha}{2}$	$\frac{1+\alpha}{2} - \alpha$	α	
$\gamma + \omega + \alpha$	γ	ω	α
	γ	ω	α

Table 3. DIRK3 Butcher Tableau
$\alpha = 0.435866521508459$
$\gamma = -\frac{6\alpha^2 - 16\alpha + 1}{4}$
$\omega = \frac{6\alpha^2 - 20\alpha + 5}{4}$

Remarks. *DIRK1 is equivalent to BDF1, the standard Backward Euler scheme. For this reason, BDF1 is dropped from the study and only the DIRK formulation of Backward Euler will be considered.*

A more convenient DIRK formulation follows from defining $\bar{\mathbf{u}}_i \doteq \mathbf{u}^{(n)} + \sum_{j=1}^{i} a_{ij}\mathbf{k}_j$. With this definition, we arrive at an alternate formulation for the DIRK scheme

$$\mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \Delta t \sum_{j=1}^{s} b_j \mathbb{M}^{-1}\mathbf{r}(t_n + c_j\Delta t, \bar{\mathbf{u}}_j)$$

$$\mathbb{M}\bar{\mathbf{u}}_i = \mathbb{M}\mathbf{u}^{(n)} + \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{r}\left(t_n + c_j\Delta t, \bar{\mathbf{u}}_j\right). \tag{15}$$

With this formulation of the DIRK scheme, $\bar{\mathbf{u}}_i$ is an approximation to the solution of (8) at time $t_n + c_i\Delta t$. This provides a more convenient and intuitive framework for predicting the solution of the nonlinear system of equations based on previous time steps in an attempt to accelerate convergence of the nonlinear solver by providing an initial guess "close" to the true solution.

III.B. Newton-GMRES Solvers

In this work, we only consider Newton-GMRES nonlinear solvers, see Algorithm 1. The standard Newton-GMRES algorithm for solving the nonlinear systems arising from the implicit discretization of a system of ODEs involves linearizing the nonlinear system at every iteration and iteratively solving the resulting linear system via GMRES [11, 12] to take a step toward the solution. At every time step, the Newton iterations are initialized with the solution at the previous time step (i.e. constant prediction). Several enhancements to the standard Newton-GMRES algorithm can be made such as better predictions for the starting point for Newton's method and Jacobian recycling techniques, which we will investigate below. Another important component of the Newton-GMRES algorithm is the convergence criteria for GMRES. A common convergence criteria when using GMRES to solve the linear system $Ax = b$ is

$$||Ax - b||_2 \leq Gtol\, ||b||_2\,, \tag{16}$$

which is the convergence criteria adopted here.

Algorithm 1 Standard Newton-GMRES Algorithm

Input: Newton Prediction: $\hat{\mathbf{u}} = \mathbf{u}^{(n)}$, GMRES Tolerance: $Gtol$, Newton Tolerance: $Ntol$
Output: Solution $\mathbf{u}^{(n+1)}$ of $\mathcal{R}_n(\mathbf{u}^{(n+1)}) = 0$
1: $\mathbf{u}^{(n+1)} = \hat{\mathbf{u}}$
2: **while** $\left|\left|\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)})\right|\right|_\infty > Ntol$ **do**
3: Evaluate residual $\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)})$
4: Evaluate Jacobian $\frac{\partial \mathcal{R}_{n+1}}{\partial \mathbf{u}}(\mathbf{u}^{(n+1)})$
5: Solve $\frac{\partial \mathcal{R}_{n+1}}{\partial \mathbf{u}}(\mathbf{u}^{(n+1)})\mathbf{p} = -\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)})$ for $\mathbf{p}$ using GMRES with tolerance $Gtol$ in (16)
6: $\mathbf{u}^{(n+1)} = \mathbf{u}^{(n+1)} + \mathbf{p}$
7: **end while**

III.B.1. Predictors for Newton's Method

In Algorithm 1, Newton's method is initialized with the solution at the previous time step. Physically, this choice of initial Newton iterate is interpreted as assuming the solution is constant between the two time steps. Unless the time step is extremely small, this is a poor assumption. Since Newton's method acts as a correction step for this prediction, there is no loss in accuracy by using a poor predictor (assuming the nonlinear solver converges), but a larger number of Newton iterations may be required. Using a cheap, accurate predictor, there is potential to speedup Algorithm 1 by reducing the number of Newton iterations per time step. Let us consider the extreme cases: If the solution of $\mathcal{R}_{n+1}(\mathbf{u}^{(n+1)}) = 0$ is chosen, then the loop will never be entered and Algorithm 1 will take essentially no time; if a very poor predictor is chosen, Algorithm 1 is not likely to converge since Newton's method is not globally convergent.

In this work, we consider two classes of predictors based on polynomial extrapolation. The first class of predictors are the Lagrange polynomials, where a k^{th} order polynomial is fit to the solution at $k+1$ points and extrapolated to generate the predictor. At step n, we write the k^{th}-order Lagrange polynomial as

$$L(t) = \sum_{j=0}^{k} \mathbf{u}_{n-j} l_j^k(t) \tag{17}$$

where

$$l_j^k(t) = \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{t - t_{n-m}}{t_{n-j} - t_{n-m}}. \tag{18}$$

where $l_0^0(t) \doteq 1$. For the BDF schemes, the predictor for time t_{n+1} is

$$\hat{\mathbf{u}} = L(t_{n+1}).$$

For the DIRK schemes, the predictor for stage j is

$$\hat{\mathbf{u}}_j = L\left(t_n + c_j \Delta t\right).$$

In this document, we only consider the constant ($k = 0$), linear ($k = 1$), and quadratic ($k = 2$) Lagrange predictors, which we denote as LAG0, LAG1, and LAG2, respectively. The other class of predictors considered are the Hermite polynomials, where a $2k - 1^{th}$ order polynomial is fit to the solution and derivative at k points and extrapolated to generate the predictor. At step n, we write the $2k - 1^{th}$ order Hermite polynomial and its derivative as

$$H(t) = \sum_{j=0}^{2k-1} \mathbf{a}_j (t - t_n)^j \tag{19}$$
$$H'(t) = \sum_{j=1}^{2k-1} j\mathbf{a}_j (t - t_n)^{j-1}$$

where the coefficients $\mathbf{a}_j$ can be determined from solving the equations:

$$H(t_{n-j}) = \mathbf{u}_{n-j} \qquad j = 0, \ldots, k$$
$$H'(t_{n-j}) = \dot{\mathbf{u}}_{n-j} \qquad j = 0, \ldots, k.$$

Define

$$\mathbf{T} = \left[\begin{array}{ccccc} 1 & 0 & 0 & \cdots & 0 \\ 1 & (t_{n-1} - t_n)^1 & (t_{n-1} - t_n)^2 & \cdots & (t_{n-1} - t_n)^{2k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (t_{n-k+1} - t_n)^1 & (t_{n-k+1} - t_n)^2 & \cdots & (t_{n-k+1} - t_n)^{2k-1} \\ \hline 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 2(t_{n-1} - t_n) & \cdots & (2k-1)(t_{n-1} - t_n)^{2k-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 2(t_{n-k+1} - t_n) & \cdots & (2k-1)(t_{n-k+1} - t_n)^{2k-2} \end{array}\right],$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0^T \\ \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_{2k-1}^T \end{bmatrix}, \qquad \mathbf{U} = \left[\begin{array}{c} \mathbf{u}_n^T \\ \vdots \\ \mathbf{u}_{n-k+1}^T \\ \hline \dot{\mathbf{u}}_n^T \\ \vdots \\ \dot{\mathbf{u}}_{n-k+1}^T \end{array}\right],$$

then the coefficients are determined from solving the $ndof$ linear systems of size $2k - 1$

$$\mathbf{TA} = \mathbf{U}. \tag{20}$$

For the BDF schemes, the predictor for time t_{n+1} is

$$\hat{\mathbf{u}} = H(t_{n+1}).$$

For the DIRK schemes, the predictor for stage j is

$$\hat{\dot{\mathbf{u}}}_j = H\left(t_n + c_j \Delta t\right).$$

Only the linear ($k = 1$), cubic ($k = 2$), and quintic ($k = 3$) Hermite predictors are considered, which are denoted HERM1, HERM2, HERM3, respectively. Notice that $\mathbf{T} \in \mathbb{R}^{2k-1 \times 2k-1}$, where $2k - 1 \leq 5$ for our problems (and most realistic applications since extrapolation using high-order polynomials is not usually advised due to potential oscillatory phenomena, especially for problems that lack smoothness). Accorindly, the matrix $\mathbf{T}$ can be factorized or inverted very cheaply, reducing (20) to $ndof$ matrix-vector multiplications of size at most 5. This is a very inexpensive operation and is negligible in comparison to a single Jacobian evaluation and linear system solve with GMRES.

Remarks. *The HERM1 predictor is equivalent to using a single step of Forward Euler as a predictor.*

III.B.2. Jacobian Recycling

Since the Jacobians arising from DG discretizations are very large, the CPU time required to perform a Jacobian evaluation (Algorithm 1, Line 4) is non-trivial, generally at least an order of magnitude more expensive than a residual evaluation (Algorithm 1, Line 3), for practical problems. Therefore, we consider Jacobian recycling methods in an attempt to speedup Algorithm 1. The result is slower convergence, i.e. more nonlinear iterations to obtain the solution, but cheaper iterations [13]. For nonlinear problems, there will be a limit to the number of times a Jacobian can be used while still generating decent inexact-Newton directions. Clearly, there are a number of different strategies that could be employed to determine the extent of Jacobian recycling and various metrics that could be used for determining when Jacobian re-computation is necessary. In this work, we reuse a Jacobian until the resulting Newton step fails to reduce the nonlinear residual at a given time step, in which case, the step is rejected, the Jacobian is recomputed at the appropriate state of the solution, and the proper Newton step is computed.

III.B.3. GMRES Tolerance

The GMRES tolerance, *Gtol*, is an important parameter in Algorithm 1 since Line 5 constitutes a significant portion of the cost in standard Newton-GMRES algorithms. When *Gtol* is small, the linear system will be solved to high precision and the computed search direction will closely match the exact Newton step. When the current iterate is near the solution of the nonlinear system, this will not impede the quadratic convergence of Newton's method. Accordingly, the number of nonlinear iterations may be small, but many GMRES iterations per nonlinear iteration may be required. The larger *Gtol*, the farther the computed step is from the true Newton direction, which may result in slower convergence of the nonlinear solver. The result is a larger number of nonlinear iterations per time step, but each nonlinear iteration is cheaper due to the relaxed tolerance on GMRES convergence.

IV. Results

In this section, we perform four numerical experiments to gain insight into the effect of ODE solver, Newton prediction, Jacobian recycling, and GMRES tolerance on accuracy and total computational time. Due to the randomness involved in timing simulations run on a shared-used cluster, the timings from these studies were obtained as follows. First, the components of a DG-FEM simulation responsible for most of the CPU time were identified. For DG problems using Newton-GMRES solution methods, these are residual evaluations, Jacobian evaluations, mass matrix evaluations, products of the mass matrix with a vector, products of the inverse of the mass matrix with a vector, GMRES iterations, and the formation of the preconditioner (Incomplete LU decomposition with no fill-in, ILU(0) [12]). Representative timings are determined for each component by executing the component several times during low-volume periods on the cluster and computing averages. Finally, each simulation is run and the number of instances of each component is counted. CPU time is easily reconstructed in a post-processing phase from the representative timings and tally of each component. This strategy ensures that any inefficiency due to machine load is felt equally by all methods.

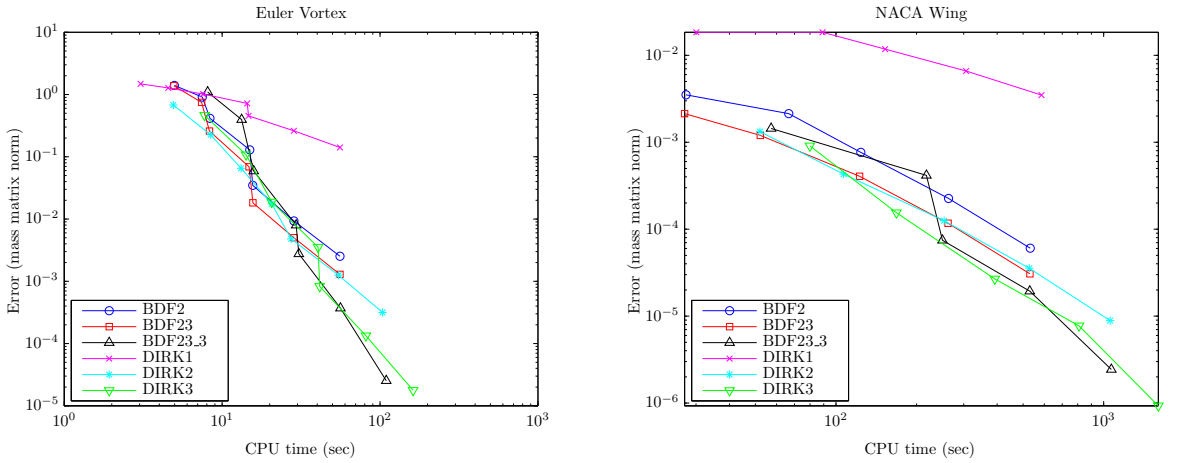
The focus of this work is ODE error, i.e. temporal error only. Accordingly, the "exact" solution for each model problem is computed by running a simulation using the DIRK3 time integration scheme with a fine time step, i.e. the most accurate scheme considered with a small time step. This ensures that the spatial error is *constant* in all convergence plots. For the remainder, the "exact" solution computed in this manner will be denoted $\mathbf{u}_e(t)$. All errors presented in this document are approximations (on the mesh of the given problem) of the absolute, continuous L2 norm of the difference between the computed solution and the "exact" solution at the final time step, i.e.

$$e^2 = (\mathbf{u}(t_0) - \mathbf{u}_e(t_0))^T \mathbb{M} (\mathbf{u}(t_0) - \mathbf{u}_e(t_0)) \approx \int_\Omega |\underline{u}(t_0) - \underline{u}_e(t_0)|^2 \, d\Omega, \tag{21}$$

where $\underline{u}$ and $\underline{u}_e$ are the continuous counterparts of $\mathbf{u}$ and $\mathbf{u}_e$, respectively.

IV.A. Experiment 1: Time Integration Scheme

In this section, we explore the effects of time integration scheme on simulation accuracy and computation time. To keep the number of figures in this document to a minimum Figure 3 shows the Pareto plots for



LAG2, Jacobian Re-computation, $Gtol = 10^{-5}$ LAG2, Jacobian Re-computation, $Gtol = 10^{-5}$

Figure 3. Comparison of Time Integration Schemes for Fixed Predictor and GMRES Tolerance

different time integration schemes for a *fixed* Newton predictor and GMRES tolerance using Jacobian re-computation. The curves are generated by varying Δt logarithmically in the range $[10^{-3}, 1]$ for the Euler vortex simulation (left) and $[3 \times 10^{-4}, 10^{-2}]$ for the NACA wing simulation (right). The plots in Figure 3 bring several conclusions to light: (1) BDF2, BDF23, and DIRK2 have nearly identical slopes for all values of Δt (as expected since all are second-order schemes) and DIRK2 contains the smallest error constant amongst the three, with BDF23 in a close second place and (2) BDF23_3 attains at least third-order convergence as expected for Δt small and has a better Pareto front than DIRK3 for small Δt . These observed trends were not significantly affected by varying Newton predictor, GMRES tolerance, or using Jacobian recycling.

IV.B. Experiment 2: Newton Predictor

In this section, the effect of Newton predictor on CPU time is studied. Assuming the predictor is sufficiently accurate as not to prevent convergence of Newton's method, the accuracy should be unaffected by varying the predictor. Figure 4 presents the results of various Newton predictors for the BDF23 (top), BDF23_3 (middle), and DIRK3 (bottom) time integrators for both the Euler vortex (left) and NACA wing (right) simulations. From these plots, it is clear that constant prediction (LAG0) leads to poor convergence for Newton's method and requires more CPU time for a given level of error. Also, particularly for the NACA wing, the quintic Hermite predictor (HERM3) is a very poor predictor due to the fact that it amounts to extrapolating a fifth-order polynomial to approximate the solution of the Navier-Stokes equations. The chaotic nature of the equations during flow separation results in an overfit causing large prediction errors and Newton's method to fail in many cases. The remainder of the predictors all perform (nearly) equally well with LAG2 being superior for most cases. These trends hold when Jacobians are recycled and for the other GMRES tolerances considered.

Finally, it can also be seen from Figure 4 that more accurate predictors are more beneficial for the BDF schemes than the DIRK scheme. A possible reason that the DIRK schemes, particularly the higher order ones, do not benefit much from accurate Newton prediction is that the individual stage values are low-order, with the high-order nature of the scheme resulting from an error cancelling linear combination of these stages. Predicting low-order stage values with highly accurate predictors may not provide a good starting point for Newton's method, causing the speedup potential of Newton prediction to be lost. In contrast, the solution of the nonlinear system of equations in the BDF schemes corresponds to solutions of the ODE. Therefore, for high-order BDF schemes, accurate Newton predictors are beneficial since they are attempting to predict an accurate solution. This observation extends to the other schemes considered in this document, BDF2, DIRK1, and DIRK2 (plots not included).

IV.C. Experiment 3: Jacobian Recycling

In this section, the effect of Jacobian recycling on CPU time is studied. Again, assuming convergence of Newton's method, the accuracy of a given scheme should be unaffected by using Jacobian recycling. Figure 5

presents the trade-off between Jacobian re-computation and recycling for the BDF23_3 time integration using LAG2 predictor with a GMRES tolerance of 10^{-5}. For small values of Δt with accurate Newton prediction, Jacobian recycling is very beneficial (factor of 2 - 3 faster than re-computation). For large values of Δt, Jacobian recycling is less beneficial, or even detrimental in the case of the Euler vortex, due to non-trivial changes in the Jacobian between time step and poor starting values for the nonlinear solver. Since all time steps were small for the NACA wing to achieve the desired accuracy, the cross-over point (where recycling becomes better than re-computation) is not shown on the right plot of Figure 5. These trends hold for the other time integrators, Newton predictors, and GMRES tolerances considered in this document (plots not included).

IV.D. Experiment 4: GMRES Tolerance

In this section, the effect of GMRES tolerance on CPU time is studied. Assuming convergence of Newton's method, the accuracy of a given scheme should be unaffected by using relaxed GMRES tolerances; varying the GMRES tolerance should result in a trade-off between number of Newton iterations required for convergence and the number of GMRES iterations required per Newton iteration. Figure 6 presents the Pareto fronts for various GMRES tolerances when BDF23_3 time integration using the LAG2 predictor is used to solve each of the model problems for the cases of Jacobian re-computation (top) and recycling (bottom). For the NACA wing (right), the simulations with $Gtol = 10^{-2}$ were faster than the simulations with $Gtol = 10^{-5}$; up to 30% faster in the case where Jacobians were recycled. For the Euler vortex, larger GMRES tolerances result in faster simulations than smaller GMRES tolerances, when Jacobians are recycled. When Jacobians are recomputed (top left), the results are largely inconclusive with many cross overs occurring among the curves.

IV.E. Performance Results

In this section, we present the results from the most promising cases in the numerical study in tabular form and compare to popular choices of certain parameters. Standard choices of the parameters discussed in this document are: DIRK3 integration scheme (or even higher-order implicit schemes) using LAG0 (constant) predictors with a GMRES tolerance of 10^{-5} and Jacobian re-computation [2, 14–16]. In Tables 4 - 9, the "Speedup over Base" field corresponds to the speedup factor of the indicated parameter combination over the standard choices just mentioned (DIRK3, LAG0, $Gtol = 10^{-5}$, Jacobian re-computation). Tables 4 and 7 present the results of BDF23 with LAG0, LAG2, and HERM1 predictors using Jacobian recycling with $Gtol = 10^{-5}$ for the two model problems. Tables 5 and 8 present the results of BDF23_3 with LAG0, LAG2, and HERM1 predictors using Jacobian recycling with $Gtol = 10^{-5}$ for the two model problems. Tables 6 and 9 present the results of DIRK3 with LAG0, LAG2, and HERM1 predictors using Jacobian recycling with $Gtol = 10^{-5}$ for the two model problems. Finally, by comparing Table 5 with Table 6 and Table 8 with Table 9 (the BDF23_3 tables with the DIRK3 tables), it is clear that BDF23_3 and DIRK3 perform similarly, in terms of both error and CPU time, with LAG0 prediction, while BDF23_3 is about a factor of 3 cheaper than DIRK3 when LAG2 prediction is used. This suggests that BDF23_3 is an alternate high-order scheme to DIRK3 with greater speedup potential from predictors such as LAG2.

	BDF23, LAG0, 10^{-5}	**BDF23, LAG2,** 10^{-5}	**BDF23, HERM1,** 10^{-5}
L2 Error	$3.24e \times 10^{-4}$	3.24×10^{-4}	3.25×10^{-4}
CPU Time (sec)	1.95×10^{4}	7.86×10^{3}	1.54×10^{4}
Speedup over Base	5.41	13.4	6.83

Table 4. Speedup/Error Results: Euler Vortex - BDF23, Jacobian Recycling

	BDF23_3, LAG0, 10^{-5}	**BDF23_3, LAG2,** 10^{-5}	**BDF23_3, HERM1,** 10^{-5}
L2 Error	2.95×10^{-6}	2.98×10^{-6}	2.91×10^{-6}
CPU Time (sec)	4.48×10^{4}	1.71×10^{4}	3.25×10^{4}
Speedup over Base	2.35	6.17	3.25

Table 5. Speedup/Error Results: Euler Vortex - BDF23_3, Jacobian Recycling

	DIRK3, LAG0, 10^{-5}	**DIRK3, LAG2,** 10^{-5}	**DIRK3, HERM1,** 10^{-5}
L2 Error	2.92×10^{-6}	2.92×10^{-6}	2.92×10^{-6}
CPU Time (sec)	4.80×10^{4}	4.13×10^{4}	4.63×10^{4}
Speedup over Base	2.20	2.55	2.28

Table 6. Speedup/Error Results: Euler Vortex - DIRK3, Jacobian Recycling

	BDF23, LAG0, 10^{-5}	**BDF23, LAG2,** 10^{-5}	**BDF23, HERM1,** 10^{-5}
L2 Error	6.34×10^{-5}	7.82×10^{-6}	7.90×10^{-6}
CPU Time (sec)	1.14×10^{3}	6.20×10^{2}	7.79×10^{2}
Speedup over Base	6.24	11.5	9.17

Table 7. Speedup/Error Results: NACA Wing - BDF23, Jacobian Recycling

	BDF23_3, LAG0, 10^{-5}	**BDF23_3, LAG2,** 10^{-5}	**BDF23_3, HERM1,** 10^{-5}
L2 Error	3.10×10^{-5}	3.11×10^{-7}	4.96×10^{-6}
CPU Time (sec)	2.38×10^{3}	1.25×10^{3}	1.46×10^{3}
Speedup over Base	3.00	5.73	4.89

Table 8. Speedup/Error Results: NACA Wing - BDF23_3, Jacobian Recycling

	DIRK3, LAG0, 10^{-5}	**DIRK3, LAG2,** 10^{-5}	**DIRK3, HERM1,** 10^{-5}
L2 Error	1.64×10^{-7}	1.15×10^{-7}	NaN
CPU Time (sec)	3.65×10^{3}	3.59×10^{3}	4.95×10^{2}
Speedup over Base	1.96	1.99	14.4

Table 9. Speedup/Error Results: NACA Wing - DIRK3, Jacobian Recycling

V. Conclusions

In this work, we took a close look at a subset of the many parameters that arise in the solution of a PDE using a discontinuous Galerkin discretization. We also introduced two new BDF-type schemes by K. Miller: second-order BDF23 and third-order BDF23_3. In particular, we focused on choice of time integration scheme, predictor for Newton's method, GMRES tolerance, and Jacobian recycling. We showed that there were non-trivial speedups that could be leveraged by simply choosing these parameters properly. On the model problems, it was shown that BDF23_3 is an attractive, high-order alternative to DIRK3; extrapolation using quadratic Lagrangian polynomials and the solution history generate reliable predictors

for Newton's method; Jacobian recycling saves CPU cost, particularly for small Δt; and $Gtol = 10^{-3} - 10^{-2}$ is a good choice for GMRES tolerance since it tends to save simulation time compared to tighter tolerances. Furthermore, we showed that using a BDF23_3 integration scheme with quadratic Lagrange polynomial predictions, a GMRES tolerance of 10^{-5}, and Jacobian recycling achieved a speedup of between 5 and 7 over the standard parameter choices of the DIRK3 scheme with constant predictor, a GMRES tolerance of 10^{-5}, and Jacobian re-computation. If a second-order scheme is acceptable for the application, a speedup of 11 to 14 can be attained by using BDF23 with quadratic Lagrange polynomial predictions and a GMRES tolerance of 10^{-5} with Jacobian recycling.

Acknowledgements

The first author acknowledges and thanks the support of a Department of Energy Computational Science Graduate Fellowship. The content of this publication does not necessarily reflect the position or policy of any of these supporters, and no official endorsement should be inferred.

References

[1]Persson, P.-O. and Peraire, J., "Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations," *SIAM J. Sci. Comput.*, Vol. 30, No. 6, 2008, pp. 2709–2733.

[2]Persson, P.-O., "Scalable Parallel Newton-Krylov Solvers for Discontinuous Galerkin Discretizations," *47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida*, 2009, AIAA-2009-606.

[3]Hesthaven, J. S. and Warburton, T., *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Springer, 2008.

[4]Peraire, J. and Persson, P.-O., "The compact discontinuous Galerkin (CDG) method for elliptic problems," *SIAM J. Sci. Comput.*, Vol. 30, No. 4, 2008, pp. 1806–1824.

[5]Brayton, R. K., Gustavson, F. G., and Hachtel, G. D., "A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas," *Proceedings of the IEEE*, Vol. 60, No. 1, 1972, pp. 98–108.

[6]Shampine, L. F. and Gear, C. W., "A user's view of solving stiff ordinary differential equations," *SIAM review*, Vol. 21, No. 1, 1979, pp. 1–17.

[7]Alexander, R., "Diagonally implicit Runge-Kutta methods for stiff o.d.e.'s," *SIAM J. Numer. Anal.*, Vol. 14, No. 6, 1977, pp. 1006–1021.

[8]Peraire, J. and Persson, P.-O., *Adaptive High-Order Methods in Computational Fluid Dynamics*, Vol. 2 of *Advances in Computational Fluid Dynamics*, chap. 5 – High-Order Discontinuous Galerkin Methods for CFD, World Scientific Publishing Co., 2011.

[9]Persson, P.-O., Peraire, J., et al., "The 3DG Project," `http://threedg.mit.edu`.

[10]Miller, K., "Two stage backward difference methods with A-stability and tuneable damping," To appear.

[11]Saad, Y. and Schultz, M. H., "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, Vol. 7, No. 3, 1986, pp. 856–869.

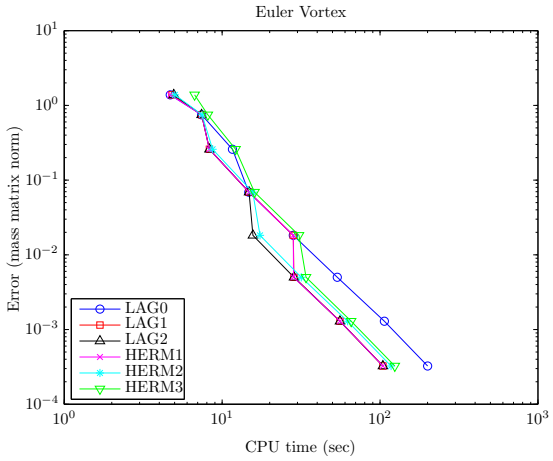
[12]Saad, Y., *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd ed., 2003.

[13]Persson, P.-O., "A sparse and high-order accurate line-based discontinuous Galerkin method for unstructured meshes," *J. Comput. Phys.*, Vol. 233, 2013, pp. 414–429.

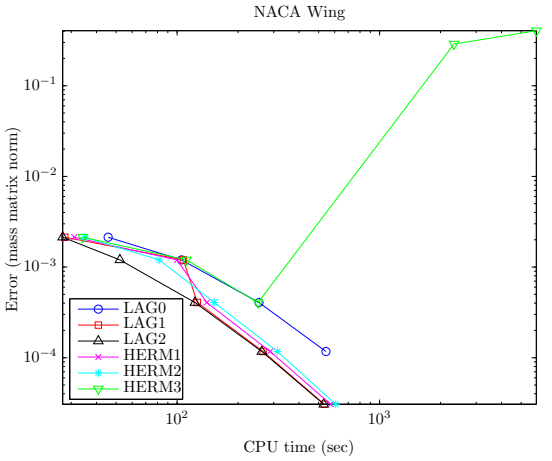
[14]Wang, L. and Mavriplis, D. J., "Implicit solution of the unsteady Euler equations for high-order accurate discontinuous Galerkin discretizations," *Journal of Computational Physics*, Vol. 225, No. 2, 2007, pp. 1994–2015.

[15]Bijl, H., Carpenter, M. H., Vatsa, V. N., and Kennedy, C. A., "Implicit time integration schemes for the unsteady compressible Navier–Stokes equations: laminar flow," *Journal of Computational Physics*, Vol. 179, No. 1, 2002, pp. 313–329.

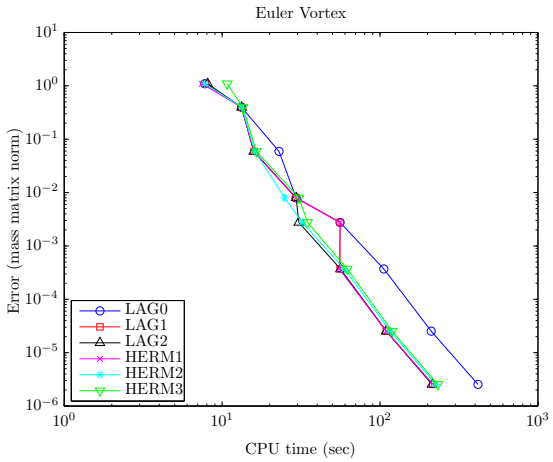
[16]Jothiprasad, G., Mavriplis, D. J., and Caughey, D. A., "Higher-order time integration schemes for the unsteady Navier–Stokes equations on unstructured meshes," *Journal of Computational Physics*, Vol. 191, No. 2, 2003, pp. 542–566.



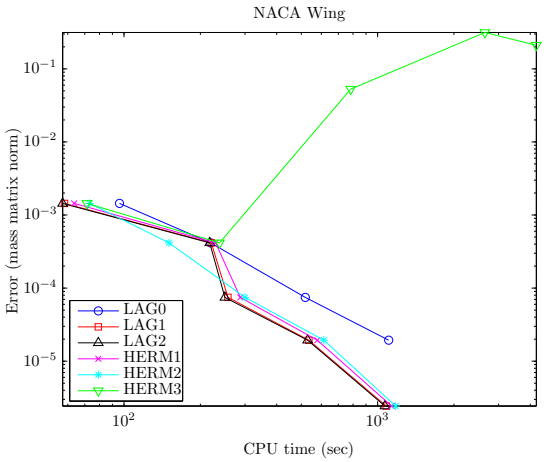
BDF23, Jacobian Re-computation, $Gtol = 10^{-5}$



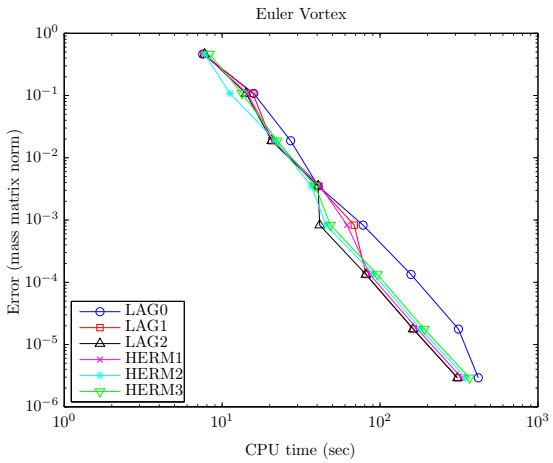
BDF23, Jacobian Re-computation, $Gtol = 10^{-5}$



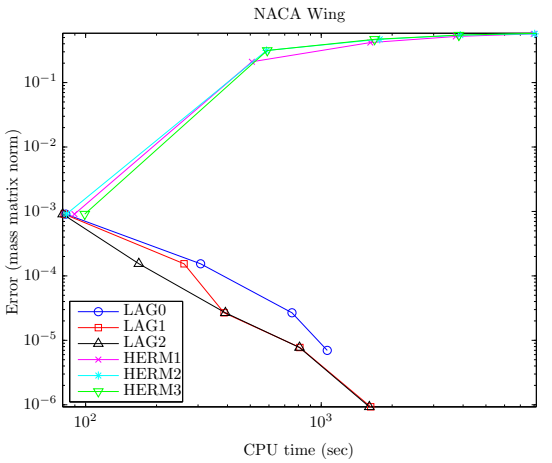
BDF23_3, Jacobian Re-computation, $Gtol = 10^{-5}$



BDF23_3, Jacobian Re-computation, $Gtol = 10^{-5}$

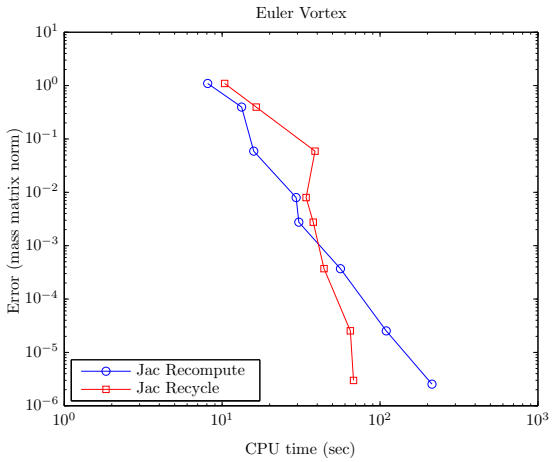


DIRK3, Jacobian Re-computation, $Gtol = 10^{-5}$

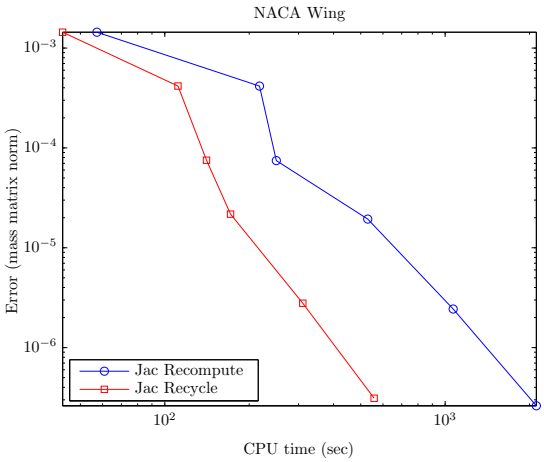


DIRK3, Jacobian Re-computation, $Gtol = 10^{-5}$

Figure 4. Comparison of Predictor for Fixed Time Integration Scheme and GMRES Tolerance

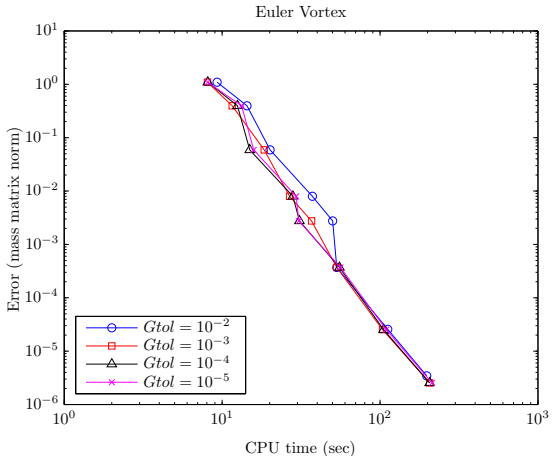


BDF23_3, LAG2, $Gtol = 10^{-5}$

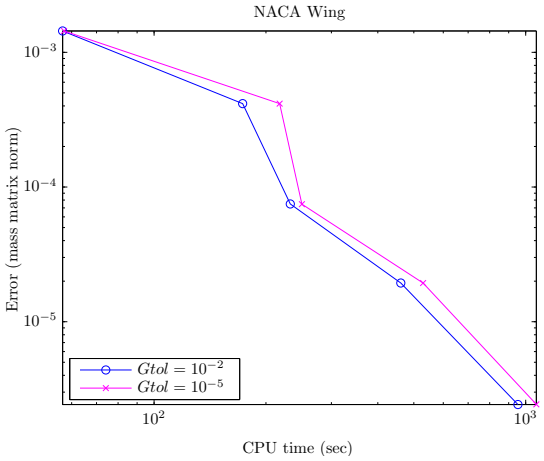


BDF23_3, LAG2, $Gtol = 10^{-5}$

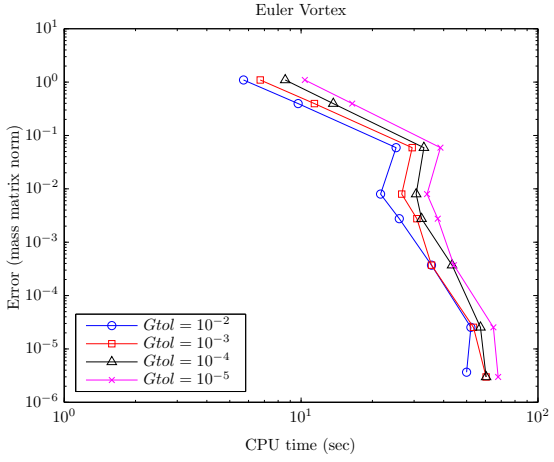
Figure 5. Comparison of Jacobian Recycling for Fixed Time Integration Scheme, Predictor, and GMRES Tolerance



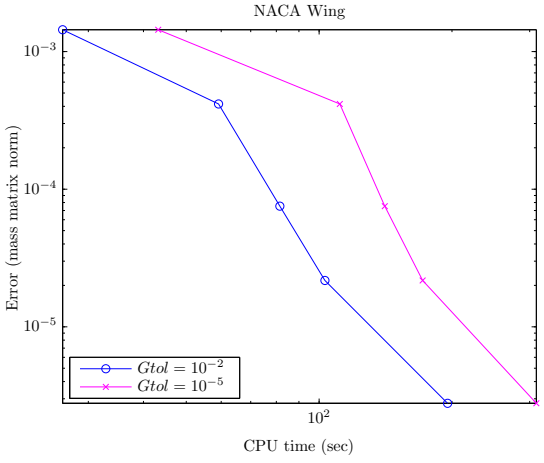
BDF23_3, LAG2, Jacobian Re-computation



BDF23_3, LAG2, Jacobian Re-computation



BDF23_3, LAG2, Jacobian Recycling



BDF23_3, LAG2, Jacobian Recycling

Figure 6. Comparison of GMRES Tolerance for Fixed Time Integration Scheme and Predictor