Oral presentation | Numerical methods

Numerical methods-VI

Thu. Jul 18, 2024 10:45 AM - 12:45 PM Room A

[10-A-04] Implicit Preconditioning for Explicit Multigrid Solvers on Cut-Cell Cartesian Meshes

*Jonathan Chiew[1], Michael Aftosmis[1] (1. NASA Ames Research Center)

Keywords: Numerical Algorithms, Computational Fluid Dynamics, Preconditioning, Multigrid

Twelfth International Conference on Computational Fluid Dynamics (ICCFD12), Kobe, Japan, July 14-19, 2024

Implicit Preconditioning for Explicit Multigrid Solvers on Cut-Cell Cartesian Meshes

J. Chiew* and M. Aftosmis*

Corresponding author: jonathan.j.chiew@nasa.gov

*NASA Ames Research Center, USA.

Abstract: This work assesses the effectiveness of linearized implicit Euler preconditioning for multigrid solvers using an unpreconditioned, Jacobian-free Newton Krylov method to converge the linear system of equations. Multigrid convergence rates improve to approximately 0.75 across the cases tested including a Mach 2 supersonic wedge, transonic NACA 0012 airfoil, and ONERA M6 wing. While larger Krylov subspaces increase the convergence rate, they also increase the computational cost, such that 4-8 Krylov vectors often offers the fastest turnaround. Further reductions in computational cost are achieved with a sequential hybrid preconditioner that begins with the explicit multigrid solver before transitioning to the preconditioned algorithm later on. In addition, a novel implementation of dual time stepping is extended to include both common BDF methods as well as high-order implicit Runge-Kutta schemes. This particular formulation, which uses A^{-1} preconditioning, is amenable to matrix-free solvers, and the L-stable methods are especially suited for meshes with arbitrarily small cut-cells. Asymptotic order of convergence is demonstrated for BDF1, BDF2, SDIRK2, and 3rd-order Radau IIA time integration with unsteady 2D vortex simulations.

Keywords: Numerical Algorithms, Computational Fluid Dynamics, Preconditioning, Multigrid.

1 Introduction

Automated and robust mesh generation and refinement are critical for simulation-based aerodynamic shape optimization, making Cartesian methods particularly attractive for arbitrarily-complex real-world vehicles. While these methods have historically been limited to inviscid flows because of the non-smooth intersection with the wetted aerodynamic surface(s), recent developments [1, 2] have extended Cartesian solvers to turbulent viscous flow. While explicit solvers have generally been sufficient for inviscid or scale-resolving simulations, state-of-the-art aircraft design and optimization typically solves the deterministic Reynolds-averaged Navier-Stokes (RANS) equations.

At high cell Reynolds numbers, converging the numerically-stiff RANS system of equations often requires implicit solvers or preconditioning. Implicit residual smoothing is often used to increase the stable CFL of an explicit multigrid method whether through a centered [3] or upwind [4] smoothing operator, accelerating convergence to steady-state. Others have investigated block-Jacobi preconditioning where the 5×5 (in 3D) cell-local flux Jacobian is inverted [5, 6] as an alternative, but for highly-stretched meshes directional coarsening or smoothing were needed to achieve the desired convergence rates [7, 8]. In order to avoid these complexities and permit even larger CFL numbers, Rossow formulated a global preconditioner based on the linearized implicit Euler method, replacing the existing residual smoothing algorithm in the multigrid method [9]. Significant convergence improvements were achieved and only a few symmetric Gauss-Seidel sweeps were needed for the linear preconditioning equations. In this work, we seek to adapt this preconditioner so that it is more suited towards our existing flow solver data structures.

Jameson [10] developed a modified version of this algorithm with similarly improved convergence rates. In addition to efficiently addressing high Reynolds number steady flows, that work showed how stronger solvers also benefitted implicit Runge-Kutta methods with their stiff, coupled stage equations. These methods have the potential to significantly increase the order of accuracy for time-dependent simulations but require the equations to be solved to much tighter tolerances at each timestep.

Considering these reasons, this work explores implicit preconditioning as a means to strengthen an existing explicit multigrid solver on unstructured Cartesian cut-cell meshes. We begin with a review of the governing equations before moving to the details of the numerical methods proposed and some computational results on a few benchmark problems from the Cart3D test suite.

2 Governing Equations

In this work, we consider three-dimensional compressible flow of an inviscid perfect gas governed by the Euler equations. For a control volume Ω enclosed by the boundary $\partial\Omega$, the integral form of these equations is given by

$$\frac{d}{dt}\iiint\limits_{\Omega} \mathbf{U} dV + \oint\limits_{\partial\Omega} \mathbf{F}\cdot\hat{n}\, dS = \mathbf{0} \tag{1}$$

where $\mathbf{U}$ is the state vector of conserved variables,

$$\mathbf{U} = [\rho, \rho u, \rho v, \rho w, \rho E]^T \tag{2}$$

$\mathbf{F}$ is the flux density tensor,

$$\mathbf{F} = \begin{bmatrix} \rho u & \rho u^2 + p & \rho uv & \rho uw & u(\rho E + p) \\ \rho v & \rho uv & \rho v^2 + p & \rho vw & v(\rho E + p) \\ \rho w & \rho uw & \rho vw & \rho w^2 + p & w(\rho E + p) \end{bmatrix}^T \tag{3}$$

ρ is the fluid density, u, v, and w are the Cartesian velocity components, p is the fluid pressure, E is the total energy per unit mass, and $\hat{n}$ is the outward-facing unit normal vector. The differential volume is $dV = dx\,dy\,dz$, and dS is the differential surface area. A simple equation of state is used to close the system of equations:

$$p = \rho(\gamma - 1)\left(E - 0.5\left(u^2 + v^2 + w^2\right)\right) \tag{4}$$

3 Numerical Methods

This work uses Cart3D, a simulation framework that solves the non-dimensionalized governing equations using the finite-volume method on multilevel Cartesian meshes with embedded boundaries [11]. Each mesh consists of regular Cartesian hexahedra throughout the computational domain, except for a layer of cells that intersect the geometric surface(s) and are arbitrarily-shaped polyhedra [12]. The spatial discretization employs a cell-centered, 2nd-order finite-volume method with a weak imposition of boundary conditions, and all results presented in this work utilize the van Leer flux function [13]. Details of temporal discretization and multigrid solver are presented in the following sections.

3.1 Steady Flow

For steady-state problems, the time-derivative term in Equation (1) is zero, leaving only the flux balance:

$$\oint\limits_{\partial\Omega} \mathbf{F}\cdot\hat{n}\, dS = \mathbf{0} \tag{5}$$

This equation defines the discrete (spatial) residual operator on the computational mesh:

$$\mathcal{R}(\mathbf{U}) = \sum_{faces} \mathbf{F}_{vL}\cdot\hat{n}\, dS = \mathbf{0} \tag{6}$$

where the subscript $_{vL}$ denotes the van Leer flux function. We begin with a description of the baseline steady-state solver and then progress to the details of the implicit preconditioning employed in this work.

3.1.1 Baseline Steady-State Algorithm

The baseline solver utilizes local timestepping (τ) to iterate Equation (6) to steady-state using a K-stage smoothing scheme with stage coefficients $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_K]$.

$$\mathbf{U}_0 = \mathbf{U}^n \tag{7}$$

$$\Delta\mathbf{U}_k = -\frac{\Delta\tau^n}{V}\mathcal{R}\left(\mathbf{U}_{k-1}\right), \qquad k = 1 \ldots K \tag{8}$$

$$\mathbf{U}_k = \mathbf{U}_0 + \alpha_k \Delta\mathbf{U}_k, \qquad k = 1 \ldots K \tag{9}$$

$$\mathbf{U}^{n+1} = \mathbf{U}_K \tag{10}$$

In Equation (8), both the local timestep ($\Delta\tau$) multiplication and division by cell volumes (V) are element-wise. The coefficients (α) are typically chosen for optimal smoothing on a single grid [14], although more complex approaches incorporating the multigrid algorithm have been developed [15]. Three and five stage schemes are commonly used (Table 1), with gradients computed only on the first stage. Both V and W multigrid cycles are supported with a recursive algorithm, and coarse meshes use a first-order spatial discretization.

	K=3	K=5
α_1	0.1481	0.0695
α_2	0.4	0.1602
α_3	1.	0.2898
α_4		0.5060
α_5		1.0

Table 1: Standard multi-stage smoother coefficients for baseline algorithm.

3.1.2 Implicitly-Preconditioned Steady-state Algorithm

Now consider implicit preconditioning of each stage update $\Delta\mathbf{U}_k$ from Equation (8), for k from 1 to K,

$$\mathcal{P}\overline{\Delta\mathbf{U}_k} = \Delta\mathbf{U}_k \tag{11}$$

$$\mathbf{U}_k = \mathbf{U}_0 + \alpha_k \overline{\Delta\mathbf{U}_k} \tag{12}$$

with $\mathcal{P}$ taken to be the preconditioner of Swanson et al. [16]:

$$\mathcal{P} = \left[\mathbf{I} + \epsilon_{\text{imp}}\frac{\Delta\tau}{V}\frac{\partial\mathcal{R}}{\partial\mathbf{U}}\right] \tag{13}$$

where ϵ_{imp} is an implicit relaxation parameter, and $\mathbf{I}$ is the identity matrix. This preconditioner can be seen as a linearized implicit Euler method, as derived by Rossow [17], or as Newton's method with pseudo-transient continuation. Rossow moved the off-diagonal Jacobian terms to the right-hand side and solved the resulting linear system with a few Symmetric Gauss-Seidel (SGS) iterations. We choose an alternative approach that does not require storing cell-face connectivity, solving Equation (12) with an approach similar to that of Xu et al. [18] utilizing GMRES [19], however, this work uses a matrix-free approach and forgoes the ILU(0) preconditioner. In order to maintain the solver's low memory requirement, we also employ the first-order Fréchet derivative to compute the Jacobian-vector product:

$$\frac{\partial\mathcal{R}}{\partial\mathbf{U}}v \approx \frac{\mathcal{R}\left(\mathbf{U}+\epsilon v\right) - \mathcal{R}\left(\mathbf{U}\right)}{\epsilon} \tag{14}$$

where the perturbation ϵ is

$$\epsilon = \sqrt{\epsilon_{\text{mach}}}\left\langle \mathbf{U}, v\right\rangle \tag{15}$$

Using an implicit preconditioner enables much larger CFL numbers which are smoothly ramped with the following equation:

$$\text{CFL}^n = \text{CFL}_{max}\tanh\left(\kappa^n\frac{\text{CFL}_0}{\text{CFL}_{max}}\right) \tag{16}$$

where CFL_0 is the starting CFL, CFL_{max} is the maximum CFL, and κ is the CFL ramping parameter.

3.2 Unsteady Flow

Many flows of interest possess significant time-dependent features like moving shocks, rotating geometry, or large regions of separated flow. One common method for simulating unsteady flow is the method of lines, where the discrete spatial residuals are evaluated along lines of constant t, resulting in a semi-discrete ODE in time. We use this framework and begin with a general review of time integration methods before discussing the specific implementation details. High order implicit schemes are of particular interest because they often require strong solvers and deeper convergence.

3.2.1 Review of Time Integration Methods

Time integration methods are often analyzed using the following scalar model equation:

$$\dot{u} = \lambda u, \qquad \lambda \in \mathbb{C} \tag{17}$$

There are two main families of time integration methods: multi-step and multi-stage. Multi-step methods store previous solutions (u) or function evaluations (λu) and combine them to advance the solution to the next timestep. In contrast, multi-stage methods compute new solutions at each stage, rather than store previous solutions (or function evaluations). These are typically computed partway into the time interval $[t, t + \Delta t]$, and then their time derivatives (i.e. function evaluations) are linearly combined to advance the solution forward in time.

Cartesian meshes with embedded boundaries may contain cut-cells with arbitrarily small volumes, which will impose serious numerical stability constraints on the timestep size with explicit methods. For this reason, we only consider A-stable implicit methods which are stable for any λ in the left half of the complex plane. Dahlquist [20] famously proved that no A-stable linear multi-step method can be greater than second-order accurate. In contrast, multi-stage methods can achieve much higher orders of accuracy while maintaining A-stability, given enough stage coefficients to satisfy the order conditions.

Researchers have developed several methods for generalizing families of time integration methods. Beam & Warming [21] developed an approach for parameterizing two-step schemes, while the General Linear form [22] essentially unifies all linear multi-step or multi-stage schemes into one formulation. In this work, we develop a framework for two-stage, implicit time integration methods described in section 3.2.3 below and highlight some of the unique aspects of our implementation.

3.2.2 Baseline Dual Time Stepping Algorithm

The baseline algorithm uses the Beam-Warming generalization of two-step linear methods:

$$(1+\xi)\,\mathbf{U}^{n+1} = (1+2\xi)\,\mathbf{U}^{n} - \xi\mathbf{U}^{n-1} + \frac{\Delta t}{V}\left[\theta\mathcal{R}\left(\mathbf{U}^{n+1}\right) + (1-\theta+\phi)\,\mathcal{R}\left(\mathbf{U}^{n}\right) - \phi\mathcal{R}\left(\mathbf{U}^{n-1}\right)\right] \tag{18}$$

although in practice only the second-order backwards difference formula (BDF2) is used because of its desirable L-stability property that ensures damping when large timesteps ($\Delta t \to \infty$) are used. For the initial startup procedure, one step of implicit Euler ($\theta = 1, \xi = \phi = 0$) is taken before switching to BDF2 ($\theta = 1, \xi = 0.5, \phi = 0$). The semi-discrete ODE is assumed to be autonomous for notational simplicity but in practice often has an explicit dependence on t when unsteady source terms, boundary conditions, or moving geometry are present. Dual time stepping [23] is used to converge the implicit equations to steady state with a secondary iteration in pseudo-time (τ), resulting in the following system of equations for BDF2:

$$V\mathcal{D}_\tau\mathbf{U} + V\frac{3\mathbf{U}^{n+1} - 4\mathbf{U}^{n} + \mathbf{U}^{n-1}}{2\Delta t} + \mathcal{R}\left(\mathbf{U}^{n+1}\right) = \mathbf{0} \tag{19}$$

where $\mathcal{D}_\tau$ is an unspecified pseudo-temporal derivative operator. Defining an unsteady residual

$$\mathcal{R}^* = V\frac{3\mathbf{U}^{n+1} - 4\mathbf{U}^{n} + \mathbf{U}^{n-1}}{2\Delta t} + \mathcal{R}\left(\mathbf{U}^{n+1}\right) \tag{20}$$

then extends a steady-state solver to time-dependent problems by simply replacing $\mathcal{R}$ with $\mathcal{R}^*$. The time derivative term is handled implicitly in the subiteration process to avoid issues with very small values of

Δt [24]. This results in the following baseline algorithm for unsteady problems integrated with BDF2:

$$\mathbf{U}_0 = \mathbf{U}^n \tag{21}$$

$$\Delta\mathbf{U}_k = -\left(1 + \frac{3\Delta\tau}{2\Delta t}\alpha_k\right)^{-1} \frac{\Delta\tau^n}{V}\left[\frac{V}{2\Delta t}\left(3\mathbf{U}_0 - 4\mathbf{U}^n + \mathbf{U}^{n-1}\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right], \qquad k = 1\dots K \tag{22}$$

$$\mathbf{U}_k = \mathbf{U}_0 + \alpha_k \Delta\mathbf{U}_k, \qquad k = 1\dots K \tag{23}$$

$$\mathbf{U}^{n+1} = \mathbf{U}_K \tag{24}$$

where the prefactor on the RHS of Equation (22) comes from the implicit handling of the time derivative term:

$$\mathbf{U}_k = \mathbf{U}_0 - \alpha_k \frac{\Delta\tau^n}{V}\left[\frac{V}{2\Delta t}\left(3\mathbf{U}_k - 4\mathbf{U}^n + \mathbf{U}^{n-1}\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right] \tag{25}$$

$$\left(1 + \frac{3\Delta\tau}{2\Delta t}\alpha_k\right)\mathbf{U}_k = \mathbf{U}_0 - \alpha_k \frac{\Delta\tau^n}{V}\left[\frac{V}{2\Delta t}\left(-4\mathbf{U}^n + \mathbf{U}^{n-1}\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right] \tag{26}$$

$$= \left(1 + \frac{3\Delta\tau}{2\Delta t}\alpha_k\right)\mathbf{U}_0 - \alpha_k \frac{\Delta\tau^n}{V}\left[\frac{V}{2\Delta t}\left(3\mathbf{U}_0 - 4\mathbf{U}^n + \mathbf{U}^{n-1}\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right] \tag{27}$$

Notice how $\mathbf{U}_0$ replaces $\mathbf{U}_k$ in the time derivative term on the RHS. This will become important for the preconditioned algorithm.

3.2.3 A^{-1}-Preconditioned Formulation for Two-stage Runge-Kutta Methods

Having detailed the baseline unsteady dual time stepping algorithm in the previous section, we now consider a new formulation to evaluate the preconditioned steady-state solver with implicit Runge-Kutta schemes. Consider a general two-stage Runge-Kutta method for the model equation:

$$u_1 = u^n + \Delta t\left(a_{11}\lambda u_1 + a_{12}\lambda u_2\right) \tag{28}$$

$$u_2 = u^n + \Delta t\left(a_{21}\lambda u_1 + a_{22}\lambda u_2\right) \tag{29}$$

$$u^{n+1} = u^n + \Delta t\left(b_1\lambda u_1 + b_2\lambda u_2\right) \tag{30}$$

The stage coefficients a_{ij} are the entries of the Runge-Kutta matrix A. As mentioned above, we consider only implicit RK methods where A is not strictly lower triangular, since explicit methods often impose severe CFL restrictions on Cartesian cut-cell meshes. In addition, we require stiff accuracy, i.e. $b_1 = a_{21}, b_2 = a_{22}$ for a more robust method. Jameson's investigation into dual time stepping with fully-implicit RK methods demonstrated that the solution update (Equation (30)), can be sensitive to the depth of convergence of $\mathcal{R}^*$. No constraints are imposed on a_{12}, so both diagonally-implicit and fully-implicit methods are considered.

Consistent with the matrix-free approach employed in this work, the stages are solved iteratively with dual time stepping. A naïve dual time stepping implementation has been shown to be unstable for small Δt [10]. In that work, preconditioning the algorithm with A^{-1} was shown to stabilize all of the schemes analyzed, including the SDIRK2 and Radau IIA methods used in this present work.

Now consider two-stage Runge-Kutta methods with A^{-1}-preconditioning in vector form using dual time stepping:

$$\mathcal{D}_\tau \overrightarrow{u_s} = \mathbf{A^{-1}}\left(\frac{\overrightarrow{u^n} - \overrightarrow{u_s}}{\Delta t} + \mathbf{A}\lambda\overrightarrow{u_s}\right) \tag{31}$$

$$= \mathbf{A^{-1}}\left(\frac{\overrightarrow{u^n} - \overrightarrow{u_s}}{\Delta t}\right) + \mathbf{I}\lambda\overrightarrow{u_s} \tag{32}$$

where $\overrightarrow{u_s}$ is the concatenated vector of stage solutions, and the vector $\overrightarrow{u^n}$ consists of multiple copies of u^N, the solution at the previous timestep. One key result of the preconditioning is highlighted explicitly in Equation (32) - choosing A^{-1} as the preconditioner decouples the stage function evaluations. This means that while iterating to convergence, the only spatial residual required is that of the *current stage*, significantly reducing the storage required. Of course the stage *solutions* are now coupled, but those are always stored and their coupling does not impact the solver memory footprint. So now each stage resembles a BDF method, with the exception that the other solutions used are converging simultaneously rather than being previously computed constants from earlier timesteps.

The A^{-1} preconditioned first stage equation with dual time stepping is now:

$$\mathcal{D}_\tau u_1 = \lambda u_1 - \frac{1}{|A|\,\Delta t}\Big(a_{22}u_1 - a_{12}u_2 - (a_{22} - a_{12})\, u^n\Big) \tag{33}$$

where $|A|$ is the determinant of the Runge-Kutta matrix. The second stage is of the same form with a change in the coefficients

$$\mathcal{D}_\tau u_2 = \lambda u_2 - \frac{1}{|A|\,\Delta t}\Big(a_{11}u_2 - a_{21}u_2 - (a_{11} - a_{21})\, u^n\Big) \tag{34}$$

and since the scheme is stiffly-accurate, $u^{n+1} = u_2$.

We can include the two common BDF methods into this framework by carefully choosing the coefficients. BDF1 is simple to include as it is a single stage, first-order DIRK method. If we examine Equation (33) and set $|A| = 1$, then we require $a_{22} = 1$ and $a_{12} = 0$ with the u^n coefficient satisfied automatically by consistency: $a_{22} - a_{12} = 1$. A simple choice for the free coefficients to give a unit determinant is $a_{11} = 1$ and $a_{21} = 0$.

For BDF2 we store the u^{n-1} solution as u_2 and again solve the Equation (33). Now $a_{22} = 1.5$, $a_{12} = -0.5$, and a nice (but non-unique) choice for the remaining coefficients is $a_{11} = a_{21} = 0.5$ which again gives $|A| = 1$. Special care must be taken to transfer $u^n \to u_2$ and $u^{n+1} \to u^n$ as the solution steps through time, but now we can compare the standard methods with more accurate IRK methods in a unified framework. For completeness, the coefficients for these two BDF schemes as well as the 2-stage IRK methods employed later in this work are given below:

$$A = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

BDF1 (1st-order accurate)

$$A = \begin{vmatrix} 1/2 & \text{-}1/2 \\ 1/2 & 3/2 \end{vmatrix}$$

BDF2 (2nd-order accurate)

$$A = \begin{vmatrix} \sqrt{2}/2 & 0 \\ 1\text{-}\sqrt{2}/2 & \sqrt{2}/2 \end{vmatrix}$$

SDIRK2 (2nd-order accurate)

$$A = \begin{vmatrix} 5/12 & \text{-}1/12 \\ 3/4 & 1/4 \end{vmatrix}$$

Radau IIA (3rd-order accurate)

While it is common practice to solve the SDIRK2 stage equations sequentially to avoid the complications of stage coupling, we solve the equations in a fully-coupled manner no matter which time integration method is utilized, giving the following update equation for a given Runge-Kutta stage:

$$\Delta \mathbf{U}_k = -\left(1 + \frac{c_1}{|A|}\frac{\Delta\tau}{\Delta t}\alpha_k\right)^{-1} \frac{\Delta\tau^n}{V}\left[\frac{V}{|A|\,\Delta t}\left(c_1\mathbf{U}_0 + c_2\mathbf{U}_s + c_3\mathbf{U}^n\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right], \quad k = 1\ldots K \tag{35}$$

where $\mathbf{U}_s$ is the most recent solution of the *other* stage and c_j are the coefficients given in Table 3.2.3. One multigrid smoothing cycle is performed for each Runge-Kutta stage every subiteration. Iterative convergence is assessed by tracking the unsteady residual ($\mathcal{R}^*$) of the second stage, since that is the advancing solution.

	Stage 1	Stage 2
c_1	a_{22}	a_{11}
c_2	a_{12}	a_{21}
c_3	$a_{22} - a_{12}$	$a_{11} - a_{21}$

Table 2: Coefficients for the two-stage A^{-1}-preconditioned dual time stepping algorithm.

3.2.4 Implicitly Preconditioned Subiteration Algorithm

We complete this section on numerical methods by describing the implicitly-preconditioned subiteration algorithm. The time integration scheme from the previous section is kept identical including the preconditioning of the dual time stepping procedure with the inverse of the Runge-Kutta matrix. In this section specifically, "preconditioning" will refer to implicit preconditioning of the subiteration updates, i.e. Equation (35), whereas "A^{-1}-preconditioning" refers specifically to the stabilization of the dual time stepping algorithm.

Consider the fully-explicit update of the dual time stepping algorithm:

$$\Delta \mathbf{U}_k = -\frac{\Delta\tau^n}{V}\left[\frac{V}{|A|\,\Delta t}\left(c_1\mathbf{U}_{k-1} + c_2\mathbf{U}_s + c_3\mathbf{U}^n\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right)\right], \quad k = 1\dots K \tag{36}$$

The baseline algorithm modifies the fully-explicit form with an implicit treatment of the time derivative term ($c_1\mathbf{U}_{k-1} \to c_1\mathbf{U}_k$). Now we consider a linearized implicit treatment of both the time derivative and spatial discretization terms

$$\Delta \mathbf{U}_k = -\frac{\Delta\tau^n}{V}\left[\frac{V}{|A|\,\Delta t}\left(c_1\left(\mathbf{U}_{k-1} + \Delta\mathbf{U}_k\right) + c_2\mathbf{U}_s + c_3\mathbf{U}^n\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right) + \frac{\partial\mathcal{R}}{\partial\mathbf{U}}\Delta\mathbf{U}_k\right], \quad k = 1\dots K$$

and then collect similar terms and add the implicit parameter ϵ as before

$$\left[\left(1 + \frac{c_1\Delta\tau^n}{|A|\,\Delta t}\right)\mathbf{I} + \epsilon\frac{\Delta\tau^n}{V}\frac{\partial\mathcal{R}}{\partial\mathbf{U}}\right]\Delta\mathbf{U}_k = -\frac{\Delta\tau^n}{V}\mathcal{R}^*\left(\mathbf{U}_{k-1}, \mathbf{U}_s, \mathbf{U}^n\right), \quad k = 1\dots K$$

where

$$\mathcal{R}^* = \frac{V}{|A|\,\Delta t}\left(c_1\mathbf{U}_{k-1} + c_2\mathbf{U}_s + c_3\mathbf{U}^n\right) + \mathcal{R}\left(\mathbf{U}_{k-1}\right) \tag{37}$$

We choose to explicitly include the dual time stepping terms on the LHS of Equation (37) rather than compute the Fréchet derivative of $\mathcal{R}^*$. This bypasses the work of adding constant terms to $\mathcal{R}^*$ that will cancel, e.g. $\mathbf{U}^n$, but still maintains a simple structure where the LHS matrix-vector product is easily evaluated combining the Fréchet derivative for the flux Jacobian product with vector additions. Note that since $\Delta\tau$ is a local timestep, the pre-factor must be computed on a cell-by-cell basis.

4 Results

In this section, we verify the preconditioned algorithms, starting with two-dimensional steady-state examples. Then the baseline and preconditioned algorithms are compared while investigating the sensitivity of the preconditioning to the input parameters before moving to a 3D problem. Finally, we conclude with an unsteady case for verification of the time integration formulation with both the baseline and preconditioned multigrid solvers.

4.1 Supersonic Wedge

We begin by simulating the flow over a supersonic wedge. Choosing $M_\infty = 2$ and a half-angle of $\delta = 15°$ results in fully supersonic flow, so the boundary states are either fully specified by the farfield state (inflow) or extrapolated from the interior (outflow). We rotate the problem so the x-direction is along the wedge surface and use a uniform 256x256 mesh with 3 levels of multigrid on the domain $x, y \in [0, 1]$.

From the classical oblique shock relations, we know that the wave angle $\beta \approx 45.3°$, so the shock should be at a 30.3° angle to the mesh. In Figure 1, we see that the shock starts at the origin and exits the domain at $y \approx \tan 30.3°$. In addition, the density jump across the shock matches the expected value of $\rho_2/\rho_1 \approx 1.729$.

Figure 2 compares the convergence of the baseline method and the preconditioned algorithm when run with $n_k = 8$ Krylov vectors, demonstrating an improved multigrid convergence rate of 0.74 from 0.88 for this problem. While this problem converges without a limiter, most realistic geometries will not unless the flow is fully subsonic, so we show convergence histories both with and without a limiter. We note that limiter rattling prevents the baseline explicit multigrid scheme from achieving deep convergence, which is typical of complex configurations, and decreases the multigrid convergence rate for both methods consistent with previous work [17].

4.2 NACA 0012 Airfoil

Next, we consider the flow around a NACA 0012 airfoil modified to have a sharp trailing edge at $M_\infty = 0.8$ and $\alpha = 1.25°$. The mesh is generated with 7 adaptation cycles using Cart3D's adjoint-

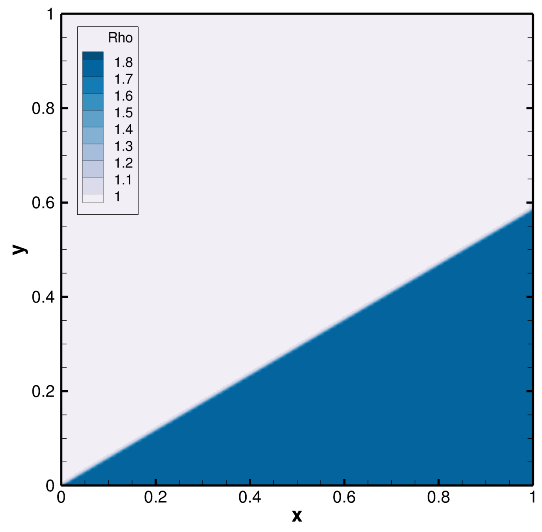

Figure 1: Density contours around a 15° supersonic wedge at Mach 2 using the van Leer limiter. The oblique shock angle and post-shock density match the analytic values of 30.3° and 1.729, respectively.

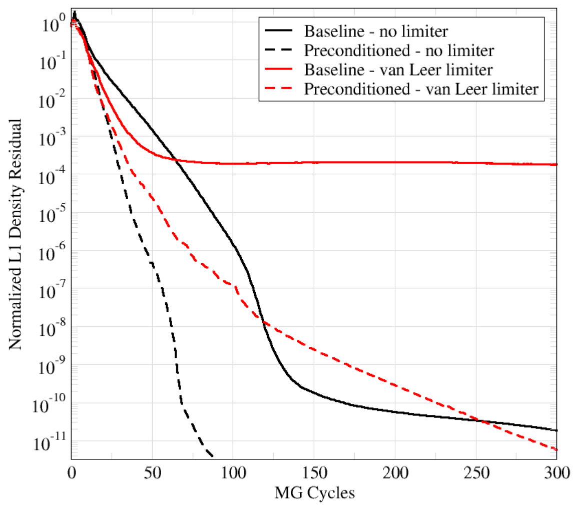

Figure 2: Comparison of iterative convergence for both the baseline (solid) and pre-conditioned (dashed) algorithms. Both algorithms show convergence degradation for this supersonic wedge case due to the van Leer limiter.

based mesh refinement capability [25] seeking to minimize the error in lift-to-drag ratio. The final adapted mesh contained ∼10k cells. One-dimensional Riemann invariant farfield boundaries were placed 40 chords away.

Figure 3 shows the adapted mesh and pressure coefficient contours around the airfoil. The mesh is much more representative of typical cases as the mesh contains regular hexahedra, 2:1 interface cells, and cut-cells along the airfoil surface, in contrast with the uniform grid used for the supersonic wedge.

The L1-norm of the density equation residual converges over 10 orders of magnitude in about 700 iterations, as demonstrated in Figure 4. After the full multigrid startup, the first few orders of magnitude are reached rapidly before the convergence rate starts to slow down. Next, we study the convergence of

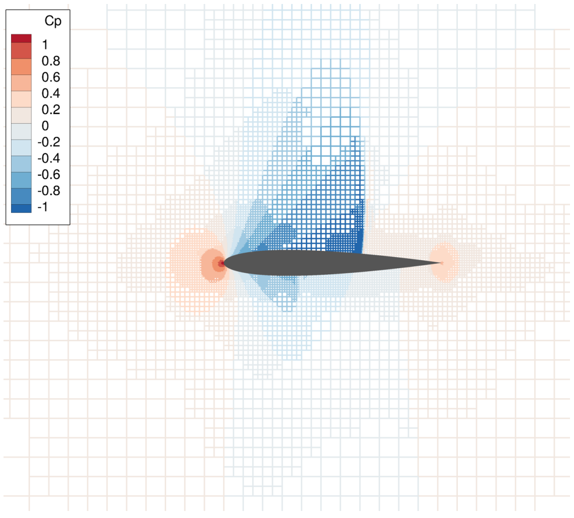

Figure 3: Adapted mesh showing pressure coefficient around a NACA 0012 airfoil at $M_\infty = 0.8$ and $\alpha = 1.25°$.

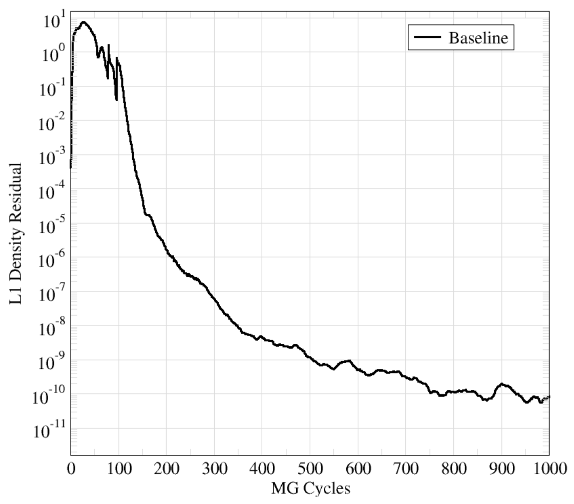

Figure 4: Iterative convergence of the NACA 0012 airfoil ($M_\infty = 0.8$, $\alpha = 1.25°$) with the baseline 5-stage, 4-level multigrid solver.

the preconditioned algorithm for the same problem. Figure 5 shows how the convergence rate improves as the number of Krylov subspace vectors is increased. There are diminishing returns after the first 6-8 vectors, which is similar to published results [9, 10, 16, 17] which used only a few SGS iterations. In this case, GMRES isn't quite as effective in the first few iterations, which is not surprising as it needs to find a solution in the basin of attraction before it starts to quickly converge. Figure 6 shows how

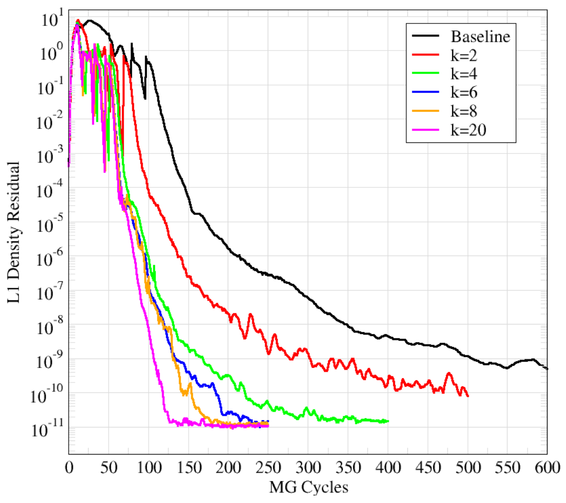

Figure 5: Iterative convergence of the preconditioned algorithm on the NACA 0012 airfoil with varying number of Krylov subspace vectors. Increasing the number of Krylov vectors improves the convergence rate but shows diminishing returns beyond 6-8 vectors.

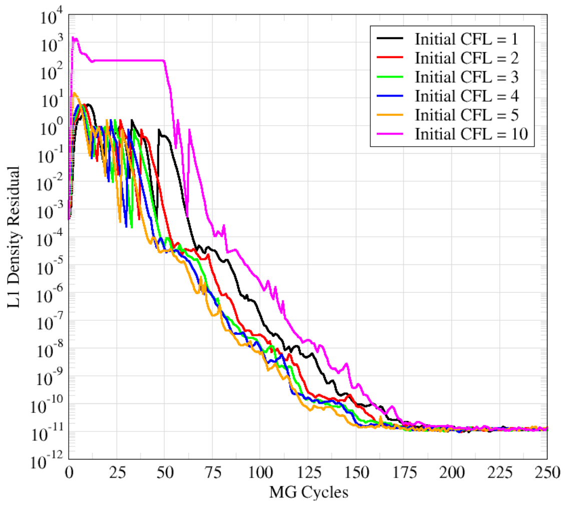

Figure 6: Iterative convergence of the preconditioned algorithm on the NACA 0012 airfoil with varying initial CFL numbers. Increasing the starting CFL up to 5 slightly reduces the number of iterations, but beyond that the coarse grids don't converge.

the convergence varies with the initial CFL number when using 8 Krylov vectors. Some reductions in iteration count to convergence are seen up to a starting CFL of 5. Beyond that, larger initial CFL numbers degrade the full multigrid startup procedure and increase the total time to solution.

Next, we study the effect of the implicit parameter ϵ in the preconditioned algorithm. Swanson et al. [16] showed how lower values had improved damping for high wave numbers, but the scheme became unstable when ϵ was too small. Figure 7 shows that this parameter has only a minor effect in the cases studied, so it is held constant in the rest of this work at $\epsilon = 0.6$.

The maximum CFL number had very little effect on the convergence history, as seen in Figure 8. Apparently, once the CFL is large enough, the flux Jacobian term of the preconditioner dominates the linear system of equations.

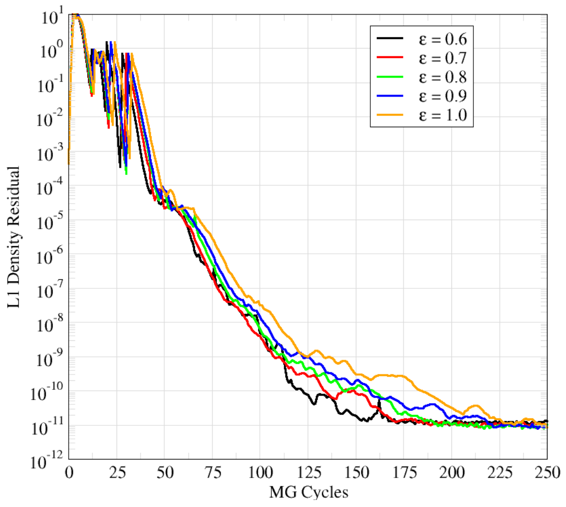

Figure 7: The implicit parameter ϵ has only a small effect on the iterative convergence for the NACA 0012 airfoil.

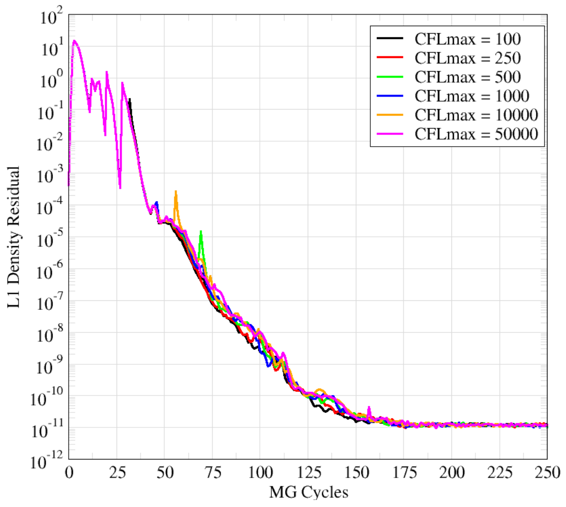

Figure 8: The maximum CFL number has almost no effect on the iterative convergence for the NACA 0012 airfoil.

Finally, we examine the effect of the CFL ramping parameter κ from Equation (16). Figure 9 shows how the hyperbolic tangent function smoothly ramps the CFL from the starting value of 5 to the maximum of 1000. At $\kappa = 2$, the CFL rapidly increases over just a few multigrid cycles, so values greater than 2 were not considered.

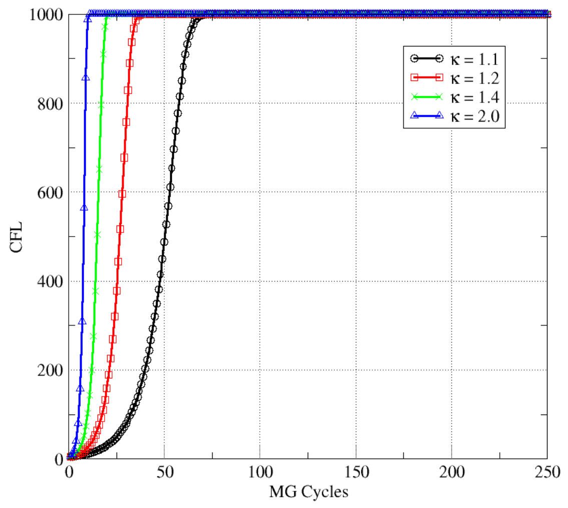

Figure 9: The CFL number is plotted for various values of the ramping parameter κ between 1.1 and 2.

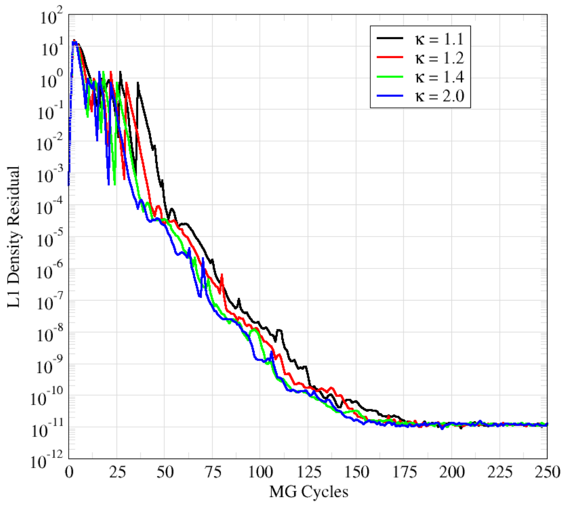

Figure 10: The CFL ramping parameter κ has only small effects on the iterative convergence for the NACA 0012 airfoil.

As one might expect, it is clear that the key parameter for the convergence rate of the preconditioned algorithm is the number of Krylov subspace vectors, N_k. Since the number of residual evaluations for the preconditioned method scales as $N_k \times N_{RK}$, using fewer Krylov vectors can be more efficient even with a degradation in the multigrid convergence rate. Based on the published literature [17], we only use 3-stage smoothing schemes with the implicit preconditioner. This is depicted in Figure 11, which

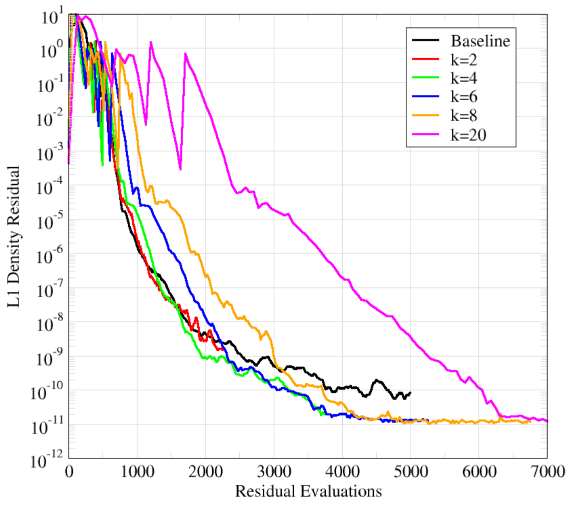

Figure 11: Convergence history plotted against total residual (flux) evaluations for various Krylov subspace sizes (N_k). The fully-explicit scheme is the most efficient unless deep convergence is required since the improved convergence rate offset by the increased computational work.

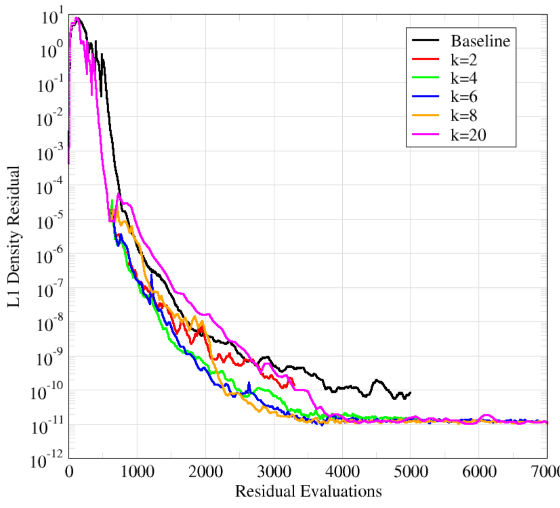

Figure 12: Convergence histories for hybrid method starting with 200 iterations of the fully-explicit scheme before enabling implicit preconditioning, demonstrating a reduction in the total number of residual evalulations for moderate Krylov subspace sizes.

shows how building a larger Krylov subspace drastically increases the amount of computational work, not even accounting for the increased number of dot products or the solution of the larger Hessenberg system. Given the faster initial convergence of explicit multigrid and better asymptotic convergence of the GMRES preconditioned solver, a natural next step would be a sequential hybrid method starting with explicit multigrid before adding in the implicit preconditioner.

Figure 12 demonstrates how this hybrid preconditioned method reduces the total number of flux evaluations needed for convergence compared to the baseline 5-stage multigrid scheme. In these runs, the first 200 iterations are performed with a fully-explicit 3-stage smoother before switching to the implicitly preconditioned scheme. One could envision further optimizing this hybrid method by automating the switching based on orders of residual reduction or convergence rate, and only after the full multigrid startup is complete.

4.3 ONERA M6 Wing

Next, we evaluate the algorithm on the ONERA M6 wing. This is a standard example case with $M_\infty = 0.54$ and $\alpha = 3.06°$. As with the NACA 0012 case, we build an adapted mesh that minimizes the error in lift-to-drag ratio. Far-field boundaries using 1D Riemann invariants are placed about 35 chords away. The pressure contours and a cutting plane through the wing root can be seen in Figure 13. After 7 adaptation cycles, the mesh is refined in areas of high pressure gradients and contains approximately 450k cells. In addition, pressure isobars are shown along the mid-span as well as near the wingtip.

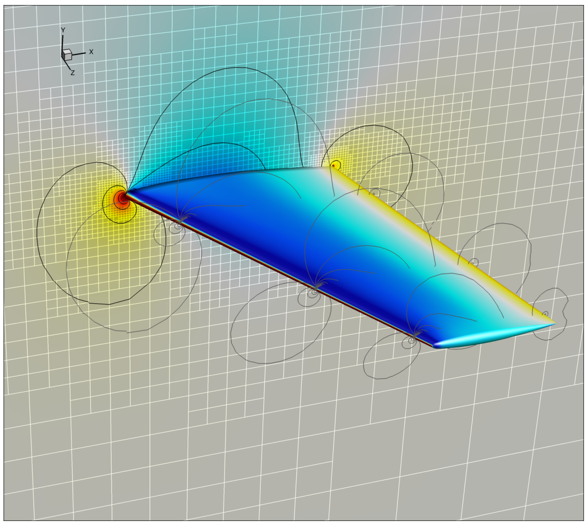

Figure 13: Pressure contours on an ONERA M6 wing at $M_\infty = 0.54$ and $\alpha = 3.06°$. A cutting plane along the wing root shows the L/D-adapted mesh. Pressure isobars are also depicted on several spanwise cutting planes.

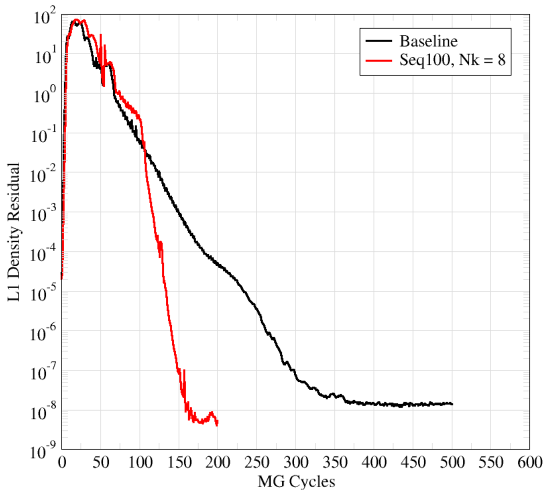

Figure 14: Convergence history for the ONERA M6 wing at $M_\infty = 0.54$ and $\alpha = 3.06°$, comparing a 5-stage explicit smoother with the sequential hybrid implicit scheme using 100 explicit startup iterations for the 3-stage smoother.

We run this case with both the baseline and implicitly preconditioned solver, setting the solver's free parameters by the results on the NACA 0012 airfoil. The same 3-stage smoothing scheme is employed with sequential preconditioning, starting with 100 explicit multigrid cycles. Figure 14 shows the improved convergence rate with the implicit preconditioning for this case with $\kappa = 1.25$, and $CFL_{max} = 1000$. When using 8 Krylov vectors, the multigrid convergence rate improves from 0.91 with the baseline algorithm to 0.75.

With this implementation, we have a developed an implicit preconditioner for the steady-state solver that significantly improves the multigrid convergence rate, without having to re-write the existing solver or its existing datastructures. The preconditioning algorithm requires storage of N_k additional state vectors for the GMRES linear solves and the computation of $N_{RK} \times N_k$ more residual evaluations. For the simple inviscid problems used in the initial evaluation, run times generally increased by 20-40%. It is likely that viscous cases with high cell Reynolds numbers and stiffer equations with a turbulence model will require the improved convergence rate of the preconditioned steady-state solver.

4.4 Vortex Convection

For time dependent flows, we consider a simple two-dimensional vortex convection problem. The flow is initialized to a Gaussian vortex centered at $(0, 0)$:

$$f = e^{g_w(1-r^2)} \tag{38}$$
$$u = M_\infty - \Gamma f y \tag{39}$$
$$v = \Gamma f x \tag{40}$$
$$T = T_\infty - g_w \frac{\gamma - 1}{\gamma} \Gamma^2 f^2 \tag{41}$$
$$p = \left(\frac{T^\gamma}{S_\infty} \right)^{\frac{1}{\gamma-1}} \tag{42}$$
$$\rho = \frac{p}{RT} \tag{43}$$

and the constants are chosen as $M_\infty = 0.5$, $\Gamma = 5/(2\pi)$, and $g_w = 0.5$. $\gamma = 1.4$ is the ratio of specific heats and $r^2 = x^2 + y^2$. One-dimensional Riemann invariant boundary conditions are applied on the x and y boundaries located $100g_w$ away and symmetry is enforced in the z-direction. The initial condition is translated by $\Delta x = -M_\infty \Delta t$ for BDF2 to alleviate the start-up problem with switching methods. Figure 15 shows the contours of density at t=0 for the Gaussian vortex.

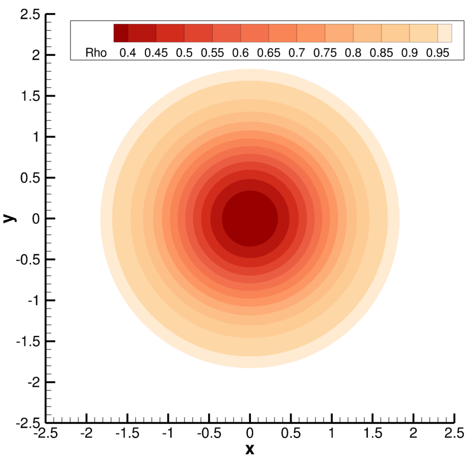

Figure 15: Density contours of the Gaussian vortex initial condition.

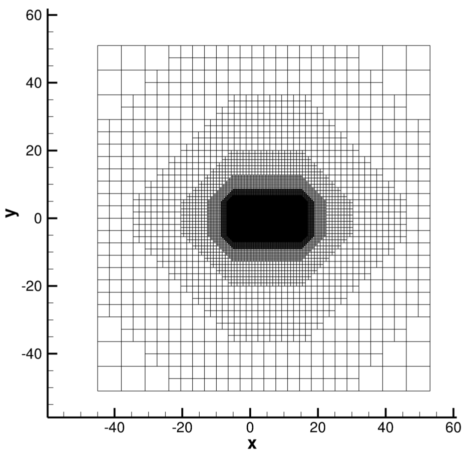

Figure 16: Computational mesh showing refinement along the vortex trajectory.

Since the spatial discretization is only 2nd-order accurate, a very fine mesh with over 35 pts per characteristic length (g_w) is used in the region $x \in (-5, 15), y \in (-5, 5)$. The entire computational domain is shown in Figure 16 and contains approximately 1.1 million hexahedra. This was chosen so that the temporal discretization error would dominate the spatial discretization errors, enabling verification of the order of convergence for each method. Even so, an extra-fine mesh with twice as many points in the x and y directions was needed for the Radau IIA simulation with the smallest Δt.

In addition to the spatial error, dual time stepping also requires the iterative error be driven below the temporal truncation error to demonstrate proper order of accuracy. For this initial verification of the implementation, the unsteady residual was converged by 6 orders of magnitude for all schemes. This was required for the implicit Runge-Kutta methods, but overly converged for the backwards difference methods. A possible extension to this approach is to estimate temporal truncation error and drive $\mathcal{R}^*$ to some factor below that tolerance, although in our experience this requires sharp error estimates for robustness.

This vortex convection case was run with all four time integration methods using the unifying formulation at several values of Δt: 0.08, 0.1, 0.125, 0.2, and 0.25. At the final time $t = 20$, the vortex has convected approximately 10 grid units downstream, and we compute the error by comparing the density along the centerline $(y = 0)$ to the exact solution over the interval $x \in [-8, 18]$. Figure 17 shows the results from these runs along with reference lines for the expected orders of accuracy. As expected, the implicit Euler solution has the most error and shows first-order convergence once Δt is in the asymptotic range. Moving to the 2nd order methods, BDF2 behaves quite similarly to BDF1 showing the largest values of Δt are outside the asymptotic range. It has more error than SDIRK2, due to its higher co-

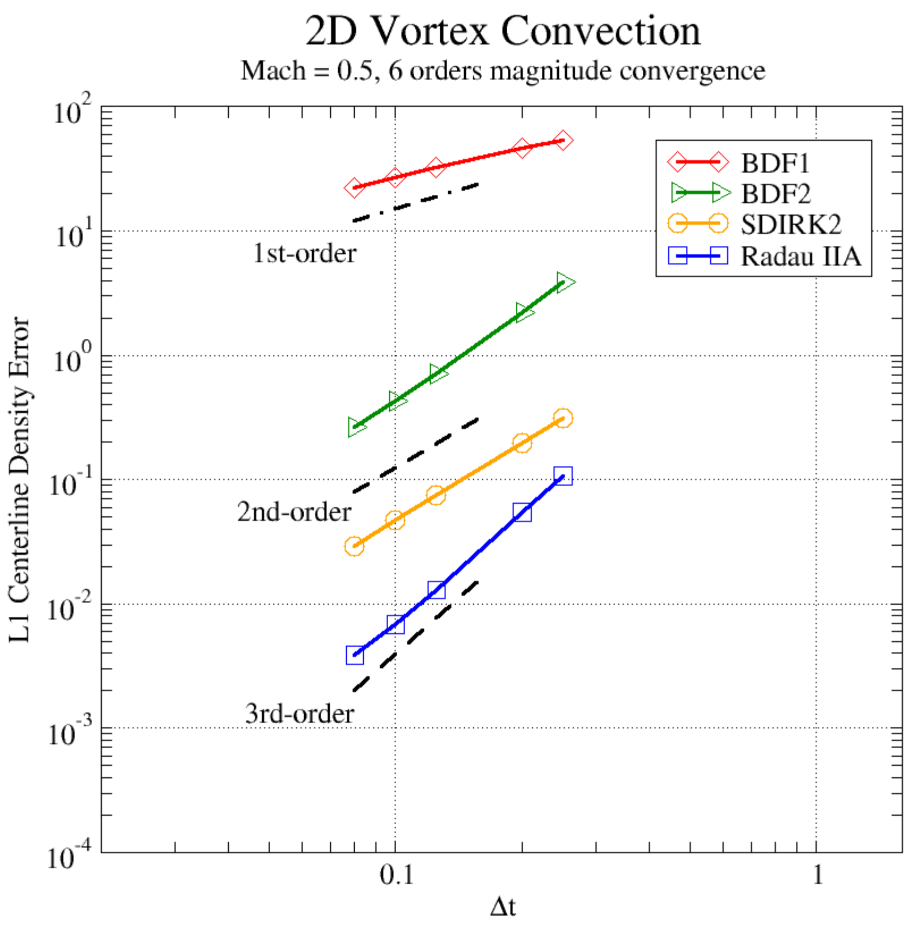

Figure 17: Order of convergence for the various time integration schemes is shown by measuring error in the centerline density (y=0) at t=20. Verification of this implementation is achieved as all methods demonstrate their asymptotic order of accuracy.

efficient on the leading truncation error term (Δt^3), but only requires the solution of a single stage at each timestep. Finally, we are able to show formal 3rd order accuracy with the Radau IIA method with significantly lower error than any of the other methods. In principle, this unified formulation also enables a p-adaptive time integration approach with 1st-3rd order L-stable methods.

5 Outlook and Future Work

In this work, we formulated an implicit preconditioner for explicit multigrid solvers on cut-cell Cartesian meshes. The linear preconditioner equations are solved with a Jacobian-free GMRES algorithm. The preconditioning provides an improvement in multigrid convergence rate on our standard test problems with as few as 2 Krylov subspace vectors. While better convergence rates are achieved with larger subspaces, they also significantly increase computational cost since the Fréchet derivatives used for the matrix-vector products require additional evaluations of the spatial residual operator on the entire mesh. Limiters are shown to significantly degrade convergence and more effort is needed to ensure realizable solutions without stalling convergence on challenging problems. We also plan to investigate the performance of this approach on an RANS solver currently in development.

In addition, we extended Jameson's implementation of dual time stepping with A^{-1}-preconditioning for fully-implicit Runge-Kutta methods. The present formulation only requires spatial residuals of the current stage and was extended to include two BDF methods. The decoupling of the residuals greatly simplifies the implementation on matrix-free solvers without significant increases in memory requirements. Asymptotic order of convergence was demonstrated for all 4 methods investigated. Further evaluation of these methods is planned to assess their computational efficiency and performance on more complicated problems.

Acknowledgments

This work was supported by the Transformational Tools and Technologies (T^3) project in NASA's Aeronautics Research Mission Directorate's Transformative Aeronautics Concepts program. Computer resources for this work were provided by the NASA Advanced Supercomputing (NAS) Division at Ames Research Center.

References

[1] Marsha Berger and Michael Aftosmis. Progress towards a Cartesian cut-cell method for viscous compressible flow. In *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2012. AIAA 2012-1301.

[2] Marsha J Berger and Michael J Aftosmis. An ODE-based wall model for turbulent flow simulations. *AIAA Journal*, 56(2):700–714, 2018.

[3] Antony Jameson. Origins and further development of the Jameson–Schmidt–Turkel scheme. *AIAA Journal*, 55(5):1487–1510, 2017.

[4] Jiri Blazek, Norbert Kroll, Rolf Radespiel, and Cord-Christian Rossow. Upwind implicit residual smoothing method for multi-stage schemes. In *10th Computational Fluid Dynamics Conference*, 1991. AIAA 1991-1533.

[5] Carl F Ollivier-Gooch. Towards problem-independent multigrid convergence rates for unstructured mesh methods I: Inviscid and laminar viscous flows. In *Proceedings of the Sixth International Symposium on Computational Fluid Dynamics*, 1995.

[6] Niles Pierce and Michael Giles. Preconditioning compressible flow calculations on stretched meshes. In *34th Aerospace Sciences Meeting and Exhibit*, page 889, 1996. AIAA 1996-0889.

[7] Niles A Pierce and Michael B Giles. Preconditioned multigrid methods for compressible flow calculations on stretched meshes. *Journal of Computational Physics*, 136(2):425–445, 1997.

[8] Dimitri J Mavriplis. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. *Journal of Computational Physics*, 145(1):141–165, 1998.

[9] Cord-Christian Rossow. Convergence acceleration for solving the compressible navier-stokes equations. *AIAA Journal*, 44(2):345–352, 2006.

[10] Antony Jameson. Evaluation of fully implicit Runge Kutta schemes for unsteady flow calculations. *Journal of Scientific Computing*, 73(2), 2017.

[11] Michael Aftosmis, Marsha Berger, and Gedas Adomavicius. A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries. In *38th Aerospace Sciences Meeting and Exhibit*, 2000. AIAA 2000-808.

[12] Michael J Aftosmis, Marsha J Berger, and John E Melton. Robust and efficient Cartesian mesh generation for component-based geometry. *AIAA Journal*, 36(6):952–960, 1998.

[13] W Kyle Anderson, James L Thomas, and Bram Van Leer. Comparison of finite volume flux vector splittings for the Euler equations. *AIAA journal*, 24(9):1453–1460, 1986.

[14] Bram van Leer, Chang-Hsien Tai, and Kenneth Powell. Design of optimally smoothing multi-stage schemes for the Euler equations. In *9th Computational Fluid Dynamics Conference*, 1989. AIAA 1989-1933.

[15] Robby Haelterman, Jan Vierendeels, and Dirk Van Heule. A generalization of the Runge–Kutta iteration. *Journal of Computational and Applied Mathematics*, 224(1):152–167, 2009.

[16] R. Charles Swanson and Cord-Christian Rossow. An efficient solver for the RANS equations and a one-equation turbulence model. *Computers & Fluids*, 42(1):13–25, 2011.

[17] Cord-Christian Rossow. Efficient computation of compressible and incompressible flows. *Journal of Computational Physics*, 220(2):879–899, 2007.

[18] Shenren Xu, David Radford, Marcus Meyer, and Jens-Dominik Müller. Stabilisation of discrete steady adjoint solvers. *Journal of Computational Physics*, 299:175–195, 2015.

[19] Youcef Saad and Martin H Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[20] Germund G Dahlquist. A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3(1):27–43, 1963.

[21] Richard Beam and Robert Warming. An implicit factored scheme for the compressible Navier-Stokes equations. II-the numerical ODE connection. In *4th Computational Fluid Dynamics Conference*, page 1446, 1979.

[22] John Charles Butcher. *Numerical methods for ordinary differential equations.* John Wiley & Sons, 2016.

[23] Antony Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. In *10th Computational Fluid Dynamics Conference*, Honolulu, HI, Jun 1991. AIAA 1991-1596.

[24] John Ming-Jey Hsu. *An implicit-explicit flow solver for complex unsteady flows.* PhD thesis, Stanford University, 2004.

[25] Marian Nemec and Michael J. Aftosmis. Adjoint error-estimation and adaptive refinement for embedded-boundary Cartesian meshes. In *18th AIAA Computational Fluid Dynamics Conference*, 2007. AIAA 2007-4187.