Eleventh International Conference on Computational Fluid Dynamics (ICCFD11), Maui, Hawaii, USA, July 11-15, 2022

ICCFD11-2022-0301

High-order temporal approaches for overset applications and AMR grids.

D. Jude, J. Sitaraman and A. Wissink
U.S. Army Combat Capabilities Development Command, Aviation & Missile Center,
Ames Research Center, Moffet Field, California, USA

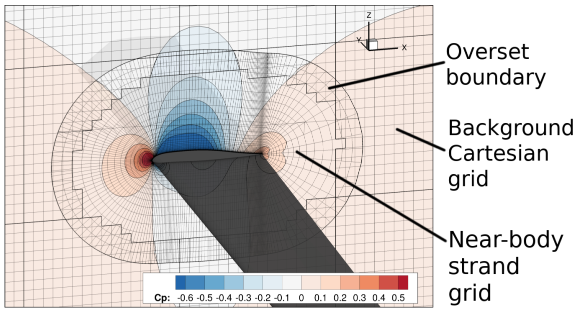
Abstract: Multi-mesh, multi-solver overset approaches for Computational Fluid Dynamics are popular for complex applications involving moving bodies. Timestepping approaches in overset CFD frameworks, however, are typically limited to 1st or 2nd order backwards differencing. Advancing the overset multi-solver paradigm to use high-order, implicit Runge-Kutta schemes involves non-trivial updates to mesh motion, domain connectivity, and flow-solver modules of an overset framework. This work investigates applying up to 4th-order implicit Runge-Kutta methods to various overset problems. First, vortex convection is studied in Cartesian meshes to verify temporal accuracy. Next a vortex-wing interaction case is studied using stationary, overset meshes. The final case involves a pitching wing mesh overset within a stationary background domain. For all overset cases usage of higher-order temporal schemes reduce the error compared to traditional 1st or 2nd order approaches.

Keywords: High-order timestepping, Numerical Algorithms, Computational Fluid Dynamics, Overset Methods.

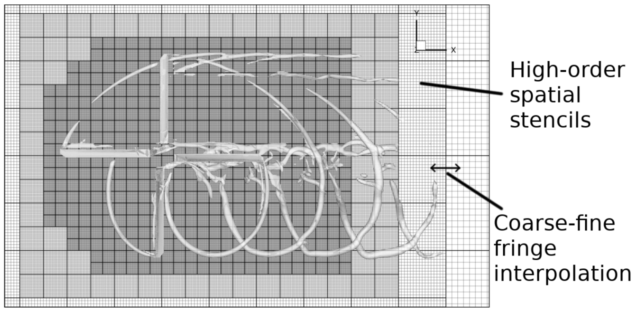
1 Introduction

Overset approaches in Computational Fluid Dynamics (CFD) have become widely adopted for simulating applications with complex geometries and moving bodies. The HPCMP CREATE$^{\text{TM}}$-AV Helios framework[1], developed for rotorcraft simulations, is a successful example of the overset, multi-solver paradigm; simulation components are meshed individually and overset within a background Cartesian Adaptive Mesh Refinement (AMR) grid.

A typical rotor simulation solution from Helios is shown in Fig. 1. The near-blade region is solved using the strand-based solver mStrand[2] and the background region is solved using the octree-based AMR solver Orchard [3]. Figure 1(a) shows the respective domains near the blade as well as the overset interface. Each solver uses its own spatial and temporal discretizations; a domain connectivity module in Helios performs hole-cutting and interpolation at domain boundaries.


Overset boundary
Background Cartesian grid
Near-body strand grid
Cp: -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 0 0.1 0.2 0.3 0.4 0.5

(a) Blade cross-section pressure



(b) Rotor Q-Criterion solution

Figure 1: Overset solution of PSP rotor using Helios with mStrand and Orchard.

Rotor simulations typically require multiple revolutions to achieve sufficient convergence of engineering quantities (rotor thrust, fuselage download, etc.). The standard for rotorcraft simulations, developed over decades with codes like Helios and NASA's OVERFLOW[4], is to use second order implicit Backwards Differencing Formulae (BDF) time-marching with an azimuthal timestep of $\Delta\psi = 0.25°$. Each revolution of the rotor takes 1440 steps and a given simulation may require anywhere from 2 revolutions for a forward flight case, to 40 revolutions for a fuselage-download problem. The BDF2 time-marching method is known to be dominated by dispersion error; this error is one of the leading hypotheses for the cause of non-physical wake breakdown in hovering rotor simulations[5].

Besides implicit BDF-type methods, explicit Runge-Kutta (RK) methods have also traditionally been used in rotor simulations. The stability of the explicit RK schemes, however, typically limit usage to background mesh regions with very small timesteps. Implicit Runge-Kutta methods on the other hand are unconditionally stable and offer high-order capability. In addition, implicit RK methods are also self-starting, making them amenable for adaptively refining time steps as a simulation progresses. Implicit-RK methods have been successfully implemented in various CFD codes[6, 7, 8, 9]. Though multi-stage implicit schemes are computationally demanding, the ability to use larger timesteps can often reduce the overall computational cost of a simulation.

The goal of this work is to demonstrate implementation of three implicit RK methods in Helios and show improved accuracy over traditional BDF methods. Three challenges from rotorcraft overset CFD applications are studied in this work as separate canonical problems. The first problem presented is isentropic vortex convection in a periodic, Cartesian domain. The second problem initializes in a Cartesian domain a line vortex which convects into an overset near-body wing mesh. The wing moment about the span-wise quarter chord is compared over time between methods. The final case considers the lift on a sinusoidal pitching wing in a background Cartesian domain.

In the broader application to rotorcraft CFD, these three canonical problems correspond to improving rotor wake resolution, capturing blade-vortex interactions, and predicting dynamic stall. Ultimately for large applications the objective is to demonstrate improved accuracy using larger azimuthal timesteps ($\Delta\psi > 0.25°$) and accelerated overall simulation times.

2 Methodology

The Helios framework consists of different modules for grid motion, overset connectivity/communication, CFD solver numerics, and rotorcraft comprehensive analysis/trim. For the purposes of studying high-order temporal schemes for overset problems, this work uses two flow solvers, mStrand and ORCHARD, the domain connectivity module PUNDIT[10], and mesh motion module MELODI[11]. mStrand is a strand-based, hybrid RANS-LES solver for near-body domains[2]. ORCHARD is an octree-based, Cartesian Adaptive Mesh Refinement (AMR), hybrid RANS-LES solver intended for background and far-field regions of a domain[3]. Each of the modules in Helios communicate through a lightweight Python infrastructure.

2.1 Governing Equations

Both mStrand and ORCHARD share the same governing equations based on the strong conservation form of the Navier–Stokes equations [12]:

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x} + \frac{\partial \mathbf{G}_i}{\partial y} + \frac{\partial \mathbf{H}_i}{\partial z} = \frac{\partial \mathbf{F}_v}{\partial x} + \frac{\partial \mathbf{G}_v}{\partial y} + \frac{\partial \mathbf{H}_v}{\partial z} + \mathbf{S} \tag{1}$$

where $\mathbf{Q}$ is the vector of conserved variables $\{\rho, \rho u, \rho v, \rho w, e\}^T$ with ρ, u, v, w and e denoting the local flow density, three components of velocity and total energy per unit volume, respectively. $\mathbf{F}, \mathbf{G}$, and $\mathbf{H}$ are the fluxes in the $x-$, $y-$, and $z-$directions and the subscripts "i" and "v" denote inviscid and viscous parts of the flux respectively. $\mathbf{S}$ is a source term which may be altered either internally in a CFD solver or externally from another code through a Python interface. The spatial residual, $\mathbf{R}$, is used for convenience to lump together all spatial operations such that

$$\frac{\partial \mathbf{Q}}{\partial t} + \mathbf{R}(\mathbf{Q}) = 0 \tag{2}$$

which is a non-linear PDE. Spatial operators for the residual term are implemented differently in each solver. The works of Lakshminarayan et. al.[2] and Jude et. al.[3] describe in more detail the convective and viscous terms for mStrand and ORCHARD respectively. For a generic Runge-Kutta multi-stage timestepping scheme the solution at the next timestep after s stages is given by:

$$\mathbf{Q}^{n+1} = \mathbf{Q}^n + h\sum_{j=1}^{s} A_{ij}k_j \tag{3}$$

where

$$k_i = -\mathbf{R}(\mathbf{Q}_i), \quad \mathbf{Q}_i = \mathbf{Q}^n + h\sum_{j=1}^{s} A_{ij}k_j, \quad h = \Delta t \tag{4}$$

and $\mathbf{A}$ is a matrix from the Butcher table, which describes the RK scheme. The Butcher table is typically written in the format

$$\begin{array}{c|c} \mathbf{C} & \mathbf{A} \\ \hline & \mathbf{B} \end{array} \tag{5}$$

where $\mathbf{C}$ is the array of time fractions, $\mathbf{A}$ is a matrix of coefficients for each stage, and $\mathbf{B}$ is the final array of coefficients for a solution update. Diagonally Implicit Runge-Kutta (DIRK) schemes are characterized by the $\mathbf{A}$ matrix being lower triangular, in which case

$$\mathbf{Q}_i = \mathbf{Q}^n + h\sum_{j=1}^{i-1} A_{ij}k_j + hA_{ii}k_i$$

$$\frac{\mathbf{Q}_i - \mathbf{Q}^n}{h} = \sum_{j=1}^{i-1} A_{ij}k_j + A_{ii}k_i = \sum_{j=1}^{i-1} A_{ij}k_j - A_{ii}\mathbf{R}(\mathbf{Q}_i) \tag{6}$$

Introducing pseudo-time term τ for pseudo-time step p and linearizing the spatial residual:

$$\frac{\mathbf{Q}^{p+1} - \mathbf{Q}^p}{\Delta\tau} + \frac{\mathbf{Q}_i^{p+1} - \mathbf{Q}^n}{h} = \sum_{j=1}^{i-1} A_{ij}k_j - A_{ii}\mathbf{R}(\mathbf{Q}_i^{p+1}) \tag{7}$$

$$\mathbf{R}(\mathbf{Q}_i^{p+1}) = \mathbf{R}(\mathbf{Q}_i^p) + \frac{\partial\mathbf{R}}{\partial q}\Delta\mathbf{Q}_i, \quad \Delta\mathbf{Q}_i = \mathbf{Q}_i^{p+1} - \mathbf{Q}_i^p \tag{8}$$

$$\left(\frac{1}{\Delta\tau} + \frac{1}{h} + A_{ii}\frac{\partial\mathbf{R}}{\partial q}\right)\Delta\mathbf{Q}_i = -\frac{\mathbf{Q}_i^p - \mathbf{Q}^n}{h} + \sum_{j=1}^{i-1} A_{ij}k_j - A_{ii}\mathbf{R}(\mathbf{Q}_i^p) \tag{9}$$

Finally if a reference state $\mathbf{Q}^*$ is defined as

$$\mathbf{Q}^* = \mathbf{Q}^n + h\sum_{j=1}^{i-1} A_{ij}k_j = \mathbf{Q}^n - h\sum_{j=1}^{i-1} A_{ij}\mathbf{R}(\mathbf{Q}_j) \tag{10}$$

then the resulting equation simplifies to the linear system

$$\left(\frac{1}{A_{ii}\Delta\tau} + \frac{1}{A_{ii}h} + \frac{\partial\mathbf{R}}{\partial q}\right)\Delta\mathbf{Q}_i = -\frac{\mathbf{Q}_i^p - \mathbf{Q}^*}{A_{ii}h} - \mathbf{R}(\mathbf{Q}_i^p) \tag{11}$$

Although the derived form of the equations are for Runge-Kutta-type schemes, the $\mathbf{Q}^*$ values for backwards-difference Euler 1st and 2nd order schemes can be derived as

$$
\begin{aligned}
\mathbf{Q}^*_{\text{BDF1}} &= \mathbf{Q}^n, \quad \mathbf{A} = [1] \\
\mathbf{Q}^*_{\text{BDF2}} &= \frac{4}{3}\mathbf{Q}^n - \frac{1}{3}\mathbf{Q}^{n-1}, \quad \mathbf{A} = [2/3]
\end{aligned}
\tag{12}
$$

and where $h = \Delta t$. When all diagonals of the **A** matrix are the same, the scheme is said to be singularly diagonal, abbreviated as SDIRK. Three SDIRK methods are implemented in Helios recommended from the work of Kennedy and Carpenter[13]: a two-stage, second order scheme (SDIRK22), a three-stage, third-order scheme (SDIRK33) and a five-stage, 4th-order scheme (SDIRK54). The Butcher table for each of the schemes is shown in Eqs. (13-15).

$$
\text{SDIRK22} \qquad
\begin{array}{c|cc}
1-\frac{\sqrt{2}}{2} & 1-\frac{\sqrt{2}}{2} & \\
1 & \frac{\sqrt{2}}{2} & 1-\frac{\sqrt{2}}{2} \\
\hline
 & \frac{\sqrt{2}}{2} & 1-\frac{\sqrt{2}}{2}
\end{array}
\tag{13}
$$

$$
\text{SDIRK33} \qquad
\begin{array}{c|ccc}
0.4358665215 & 0.4358665215 & & \\
0.7179332608 & 0.2820667392 & 0.4358665215 & \\
1 & 1.208496649 & -0.644363171 & 0.4358665215 \\
\hline
 & 1.208496649 & -0.644363171 & 0.4358665215
\end{array}
\tag{14}
$$

$$
\text{SDIRK54} \qquad
\begin{array}{c|ccccc}
\frac{1}{4} & \frac{1}{4} & & & & \\
\frac{3}{4} & \frac{1}{2} & \frac{1}{4} & & & \\
\frac{11}{20} & \frac{17}{50} & \frac{-1}{25} & \frac{1}{4} & & \\
\frac{1}{2} & \frac{371}{1360} & \frac{-137}{2720} & \frac{15}{544} & \frac{1}{4} & \\
1 & \frac{25}{24} & \frac{-49}{48} & \frac{125}{16} & \frac{-85}{12} & \frac{1}{4} \\
\hline
 & \frac{25}{24} & \frac{-49}{48} & \frac{125}{16} & \frac{-85}{12} & \frac{1}{4}
\end{array}
\tag{15}
$$

The entire right-hand-side (RHS), including the temporal terms, of Eq. (11) is referred to as the residual for time-accurate computation. The L2-norm of the residual is a measure of subiteration convergence, since as the solution at p approaches $p + 1 = n + 1$, the non-linear form of the equation is recovered. The inversion of the left-hand-side (LHS) matrix is rarely performed exactly, since this process is typically very computationally expensive. Instead each solver in the overset framework provides a preconditioner to approximate the solution $\Delta\mathbf{Q}$. The preconditioner can be used as an approximate linear-solver to advance the solution in time. A Generalized Minimal Residual (GMRES) linear solver is also implemented in Helios for more challenging problems. Interpolation and communication between meshes is performed at each linear-solver iteration within GMRES, as described in the work of Jude et. al.[14].

2.1.1 Moving and Deforming Grid Considerations

In the case of a moving or deforming grid the governing equations include grid velocity terms. The Geometric Conservation Law (GCL) is a constraint on how to include grid deformations in a way that preserves freestream conditions. Instead of considering point velocities at grid nodes, the volume swept by a cell face over the timestep should be used to obtain the face velocities required in flux computations. The works of Yang and Mavriplis describes in detail the treatment of grid velocity terms in multi-stage RK schemes[6]. In summary, and described for SDIRK33 in Eq. (16), the grid velocity of a cell face, $\dot{\mathbf{X}}$, can be obtained using the same Butcher table coefficients on the left-hand side but with the volume swept by the face at stage k, $V_k - V_n$, on the right-hand-side.

$$
\begin{pmatrix}
a_{11} & 0 & 0 \\
a_{21} & a_{22} & 0 \\
a_{31} & a_{32} & a_{33}
\end{pmatrix}
\begin{Bmatrix}
\dot{\mathbf{X}}^1 \\
\dot{\mathbf{X}}^2 \\
\dot{\mathbf{X}}^3
\end{Bmatrix}
= \frac{1}{\Delta t}
\begin{Bmatrix}
V_1 - V_n \\
V_2 - V_n \\
V_3 - V_n
\end{Bmatrix}
\tag{16}
$$

2.2 Overset Framework Description

Helios combines multiple solvers in a single framework using light-weight Python wrappers. Each CFD solver or other module is written in a mix of C, C++, and Fortran; memory pointers are then exposed to Python wrappers to enable data-sharing between Helios framework components. Any time-marching loops in Helios described in this section are driven from the Python layer using function calls and memory pointers from a compiled, fast numerical library.

Algorithm 1 Helios baseline operation for time-stepping with multiple, overset solvers.

```
1: $T \leftarrow 0$
2: for $n = 1, ..., N_{\text{steps}}$ do
3:     $T \leftarrow T + \Delta T$
4:     MOVE_GRIDS($T$)                                    ▷ Mesh Motion
5:     DOMAIN_CONNECTIVITY()                              ▷ Hole-cutting and connectivity
6:     for all $S \in$ CFD Solvers do
7:         OVERSET_INTERPOLATE($\mathbf{Q}$)
8:         $S$.RUN_STEP($n$)                              ▷ Solver advances solution
9:     end for
10: end for
```

Algorithm 2 Helios multi-stage, overset time-integration.

```
1: $T \leftarrow 0$
2: for $n = 1, ..., N_{\text{steps}}$ do
3:     $\mathbf{Q}^n \leftarrow \mathbf{Q}$
4:     $\mathbf{R}_{\text{s}} \leftarrow [\emptyset, ..., \emptyset]$          ▷ Storage for spatial residuals, 1 per stage
5:     for $i = 1, ..., N_{\text{stage}}$ do
6:         $T_i \leftarrow T + \mathbf{C}[i] \cdot \Delta T$
7:         MOVE_GRIDS($T_i$)                              ▷ Advance grid to position at step $i$
8:         DOMAIN_CONNECTIVITY()                          ▷ Hole-cutting and connectivity
9:         $\mathbf{Q}^* \leftarrow \mathbf{Q}^n$
10:        for $j = 1, ..., i-1$ do
11:            $\mathbf{Q}^* \leftarrow \mathbf{Q}^* - A_{i,j} \cdot \Delta T \cdot \mathbf{R}[j]$
12:        end for
13:        for $m = 1, ..., N_{\text{sub-iter}}$ do
14:            OVERSET_INTERPOLATE($\mathbf{Q}$)
15:            for all $S \in$ CFD Solvers do
16:                $h \leftarrow A_{i,i} \cdot \Delta T$
17:                $\mathbf{R}$, $\mathbf{R}_{\text{s}}[i] \leftarrow S$.COMPUTE_RHS($\mathbf{Q}, \mathbf{Q}^*, h$)
18:            end for
19:            $\Delta\mathbf{Q} \leftarrow$ DO_GMRES($\mathbf{R}$)
20:            $\mathbf{Q} \leftarrow \mathbf{Q} + \Delta\mathbf{Q}$
21:        end for
22:    end for
23:    $T \leftarrow T + \Delta T$
24: end for
```

In standard operation, Helios roughly follows Alg. 1 for time integration. Each CFD module is independently called per timestep to perform a non-linear solve over the system of equations and any overset communication is done once per solver, per timestep. Recently an "Overset GMRES" (O-GMRES) method was added to Helios, combining lines 6-9 from Alg. 1 in a more tightly coupled solver for the complete overset system [14], described in Alg. 3. Even with O-GMRES, however, the connectivity and grid motion are still performed once per timestep and the overall temporal accuracy is limited to BDF-type schemes like BDF1 and BDF2 in each solver.

To enable high-order time-marching with a multi-solver framework, the multi-stage time-marching scheme has been implemented at the Python level in Helios. The modified approach for a multi-stage, SDIRK scheme is shown in Alg. 2. Both mStrand and ORCHARD required minor modifications to work with an RK algorithm driven externally from Python. The COMPUTE_RHS routine in each solver, which computes the right-hand side of Eqn. 11, has two changes. First, it uses $\mathbf{Q}^*$, the reference state for the backward-differencing term. Second it returns both the full residual and spatial portion of the residual, $\mathbf{R}(\mathbf{Q}_i^p)$, so that it can be stored and used in later stages of the RK algorithm.

Grid motion routines from the MELODI module in Helios also required minor modifications to accept the solution time T_i, which is no longer a constant delta from the previous call. For certain schemes it is also not guaranteed that time increases monotonically over RK stages. The time fractions for SDIRK54, for example, decrease from 3/4 to 11/20 between the second and third stages.

Line 19 from Alg. 2 is the computationally expensive linear-solver of the implicit system of equations. The DO_GMRES function call refers to the aforementioned O-GMRES algorithm, also implemented at the Python level and described in Alg. 3 [14]. O-GMRES solves the full, overset system of linear equations associated with each (non-linear) sub-iteration for each RK-step. The norm of a single residual vector, $\mathbf{R}$, is used to measure linear and non-linear convergence at each RK stage. Not shown in Alg. 2 is a controller to check convergence and exit if a certain convergence threshold is met.

Algorithm 3 Right-Preconditioned O-GMRES Algorithm using notation consistent with the work of Saad[15]. Symbols like A, for example, do not correspond to the previously described $\mathbf{A}$ from the RK Butcher table.

```
1:  function DO_GMRES($b$)
2:      $x_o \leftarrow [0, ..., 0]$
3:      Compute $r_o = b - Ax_o$, $\beta := ||r_o||_2$, and $v_1 := r_o/\beta$
4:      for $j = 1, 2, ..., m$ do
5:          Compute $r_j :=$ PRECONDITION($v_j$)          ▷ Preconditioner (CFD solver routine)
6:          OVERSET_INTERPOLATE($r_j$)
7:          Compute $w_j := Ar_j$          ▷ Matrix-Vector Product (CFD solver routine)
8:          for $i = 1, ..., j$ do
9:              $h_{ij} := (w_j, v_i)$          ▷ Vector Dot Product
10:             OVERSET_SUM($h_{ij}$)
11:             $w_j := w_j - h_{ij}v_i$
12:         end for
13:         $h_{j+1,j} = ||w_j||_2$
14:         if $h_{j+1,j} == 0$ then
15:             $m := j$
16:             goto 20
17:         end if
18:         $v_{j+1} = w_j/h_{j+1,j}$
19:     end for
20:     Define the $(m+1) \times (m)$ Hessenberg matrix $\tilde{H} = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$
21:     Compute $y_m$, the minimum of $||\beta e_1 - \tilde{H}_m y||_2$ and $r_m = V_m y_m$
22:     Compute $x_m = x_o +$ PRECONDITION($r_m$)
23:     OVERSET_INTERPOLATE($x_m$)
24:     return $x_m$
25: end function
```

3 Results

Evaluating temporal accuracy involves running a simulation with different timestepping schemes and comparing a physical quantity of interest between the methods. An approximation of the "exact" solution can be found by taking a very small timestep with the highest-order scheme and using that as a basis for error estimation. Even for cases where a true exact solution may be known, the full error would include both temporal and spatial components. The approximated "exact" solution includes spatial errors and therefore comparison against that solution helps isolate temporal error.

ORCHARD and mStrand non-dimensionalize velocities with respect to the speed of sound. Consequently, the time t in each solver is also non-dimensionalized as

$$t = \frac{t_{\text{phys}} * a_\infty}{D} \tag{17}$$

where a_∞ is the speed of sound far in the far-field and D is the dimension of 1 grid unit. All reported times and timesteps in this results section are in non-dimensional form.

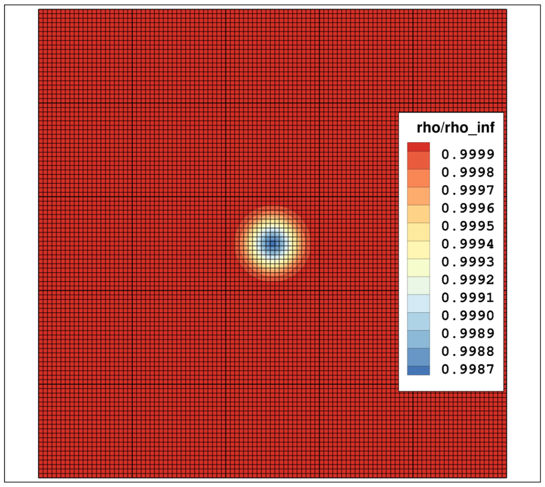
Some results refer to a "BDF2*" method, which is the BDF2 method applied separately in each solver with coupling once per timestep (Alg. 1). The BDF2* method is the current standard operation in Helios and is therefore often used as a comparison baseline for other methods.

3.1 Vortex Convection

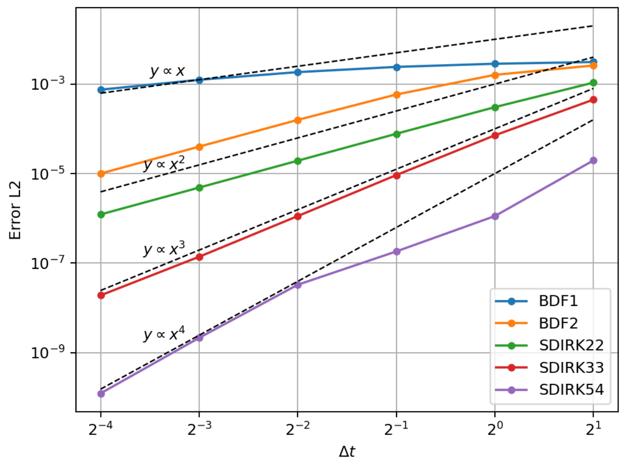
An isentropic vortex is initialized in an ORCHARD Cartesian domain with periodic boundaries and without a mStrand mesh. For a freestream Mach number of 0.2, the vortex travels a distance of 20 units in time $T = 100$. The finest cell spacing in the refined area of the domain is $\Delta x = 0.078125$. An "exact" solution is obtained by using the highest-order method, SDIRK54, with a small timestep of $\Delta t = 100/3200 = 0.03125$, corresponding to a Courant-Friedrichs-Lewy (CFL) number of 0.48. Six progressively coarser timesteps are used to compare the error against the exact solution, summarized in Table 1. The temporal residuals for each case were converged over 10 orders of magnitude to ensure any resulting error is a result of the temporal scheme. Figure 2(a) shows the initial solution and grid for the case.

Table 1: Summary of the seven cases run to analyze temporal accuracy.

Case	"exact"	1	2	3	4	5	6
Steps	3200	1600	800	400	200	100	50
Δt	0.03125	0.0625	0.125	0.25	0.5	1.0	2.0
CFL	0.48	0.96	1.92	3.84	7.68	15.36	30.72

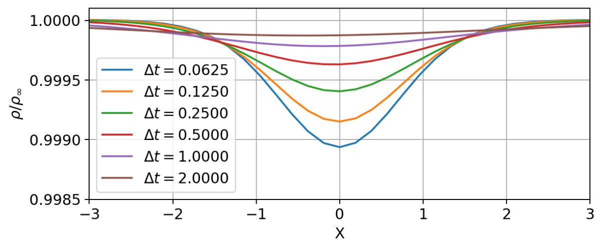


(a) Full domain and initial solution.

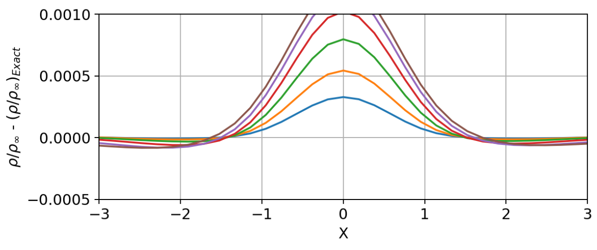


(b) Error convergence.

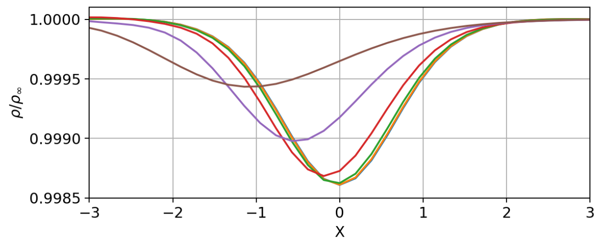
Figure 2: Vortex convection (a) initial pressure solution and (b) temporal convergence showing order of accuracy.



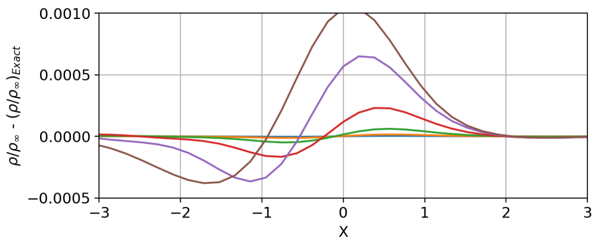
(a) BDF1 Value



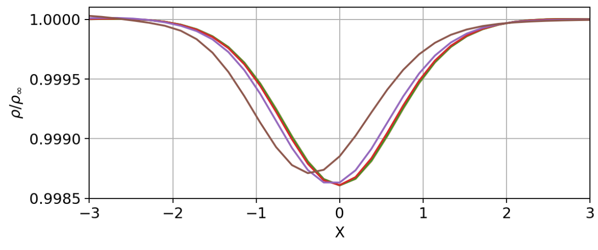
(b) BDF1 Error



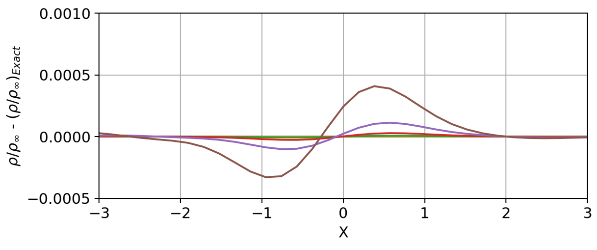
(c) BDF2 Value



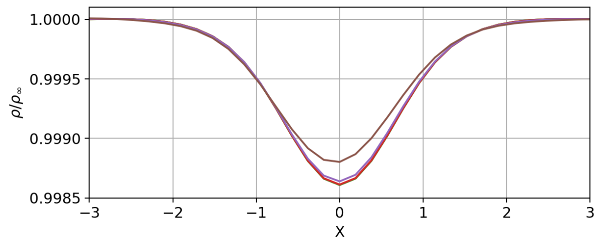
(d) BDF2 Error



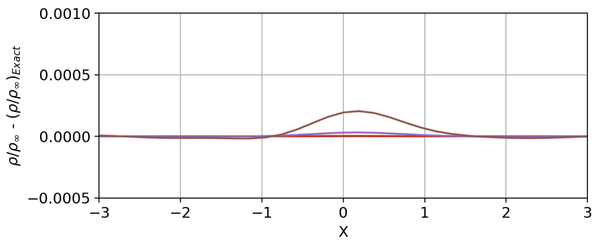
(e) SDIRK22 Value



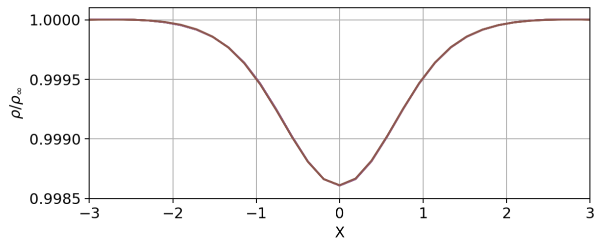
(f) SDIRK22 Error



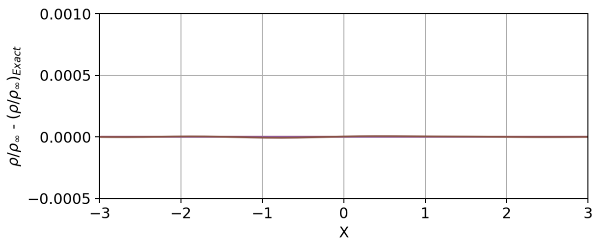
(g) SDIRK33 Value



(h) SDIRK33 Error



(i) SDIRK54 Value



(j) SDIRK54 Error

Figure 3: Density solutions and errors across the vortex center-line at time $T = 100$ (one domain traversal, 20 units distance).

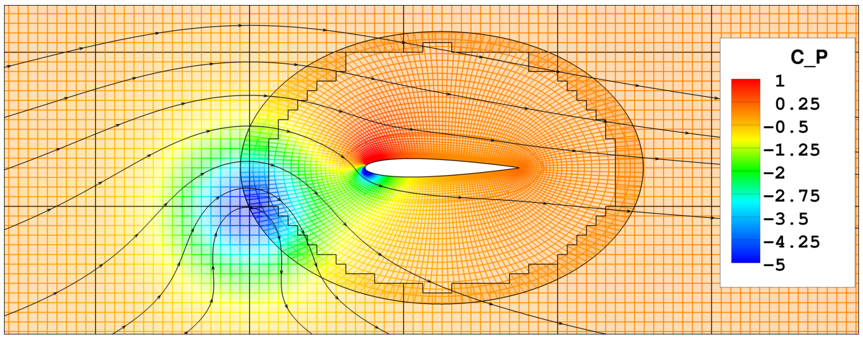
The density solution across the centerline of the vortex is extracted for each case and compared against the exact solution. Figure 3 shows density profiles and error for each of the timestepping schemes: BDF1, BDF2, SDIRK22, SDIRK33, and SDIRK54. The BDF2 results, in Fig. 3(c) and (d) show poor accuracy with even moderate timesteps; both diffusion and dispersion error dominate. SDIRK22 is also a second-order scheme however the accuracy is significantly improved; dispersion error appears to be the dominant error with SDIRK22. SDIRK33, being an odd-order scheme, shows primarily diffusion error as expected. Finally SDIRK54 shows almost no distinguishable error on the scales shown.

The error for a given scheme is determined by taking the L2 norm of the difference between the approximate and exact density distributions along the vortex center-line. The error for each case is shown in Fig. 2(b). On a log scale the slope of each error line should match the temporal order of accuracy of the scheme. The BDF1 method matches a slope-1 line at small timesteps but has an even lower slope at the larger timesteps. Both BDF2 and SDIRK22 show slope-2 convergence, though as previously noted the error for BDF2 is higher in all cases than SDIRK22. Finally the SDIRK33 and SDIRK54 schemes show slope-3 and slope-4 trends, as expected.

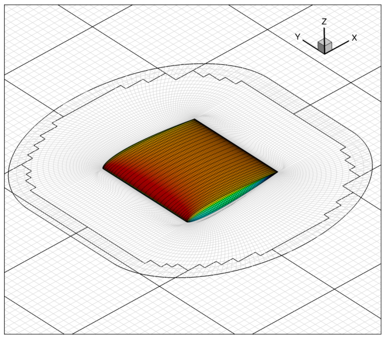
3.2 Vortex-Wing Interaction

An overset, inviscid wing-vortex interaction case was run with Orchard and mStrand. The wing has NACA0012 airfoil cross-sections, aspect ratio of 1, and unit length, as shown in Fig. 4. A line vortex is initialized in the Orchard domain 7 chords up-stream and 0.25 chords below the wing quarter-chord. The line vortex extends for the entire length of the Y- (spanwise-) direction of the domain; the Orchard domain is also set to periodic in this direction. The background domain has fixed refinement of 0.0625 chords from $x = -9$ upstream of the wing to $x = 6$ downstream of the wing. The freestream Mach number is 0.2, meaning the vortex travels 1 unit length in non-dimensional time $T = 5$. The SDIRK22 method was used initially from $T = 0$ to $T = 15$ while the vortex convects 3 chords and a physical solution develops on the wing. The simulation is then restarted from $T = 15$ with various timestepping schemes and timestep sizes.

This vortex-wing simulation is analogous to a blade-vortex interaction (BVI) case often studied in helicopter aerodynamics. The tip of a representative rotor with an aspect ratio $R = 16$ and azimuthal timestep of $\Delta\psi = 0.25°$ will travel $\pi R/720 \approx 0.07$ chords per timestep. If this corresponds to a tip Mach number of 0.6 (typical for a hovering rotor or low-speed flight), the equivalent non-dimensional timestep is $\Delta t \approx 0.12$. The BDF2 result from Fig. 6 at $\Delta t = 0.125 = 2^{-3}$ is therefore comparable to the error we expect from best-practice rotor simulation at $\Delta\psi = 0.25°$.



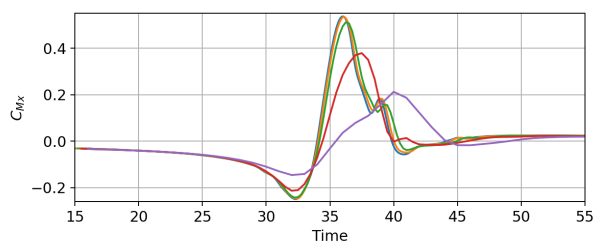
(a) Center-plane pressure solution.



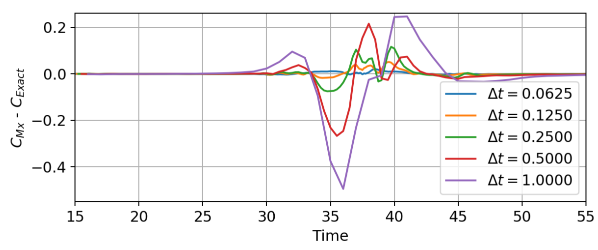
(b) Isometric view of 3d wing.

Figure 4: Solution at $T = 30$ when the vortex passes from the background to near-body mesh.

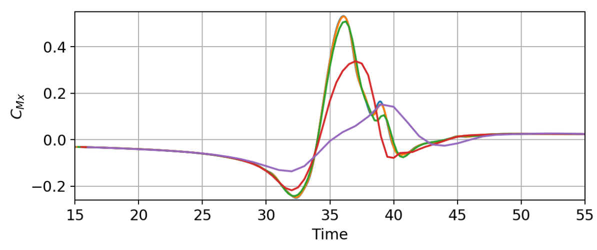
The solution at time $T = 30$ is shown in Fig. 4. The vortex at $T = 30$ has traveled 6 chords to only 1 chord upstream of the wing quarter-chord, and the vortex core can be seen entering the near-body domain. Streamlines are shown to illustrate the influence of the vortex on the flow field. The solution at $T = 30$ is quite difficult to properly converge as a result of high gradients in the interpolation region between meshes. Use of O-GMRES is required for this case to obtain sufficient convergence; with a simple Gauss-Seidel approximate linear solver the residuals from both mStrand and Orchard stall.



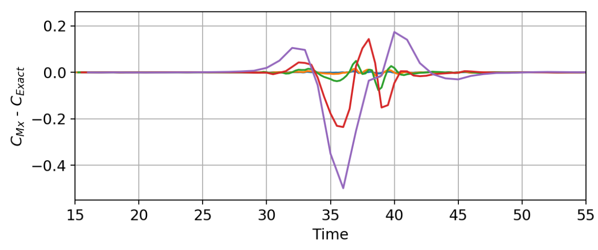
(a) BDF2* Value



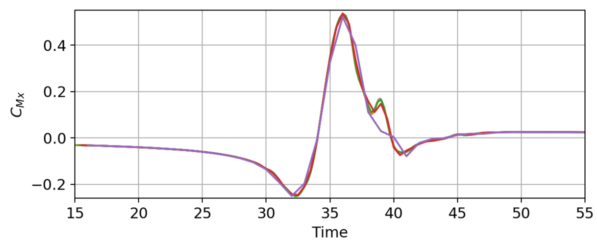
(b) BDF2* Error



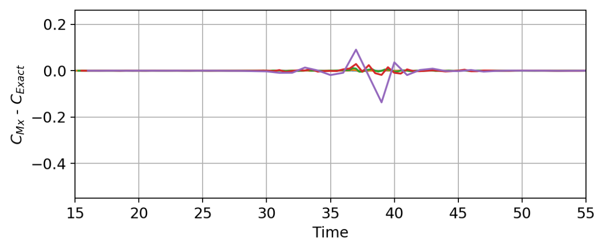
(c) BDF2 Value



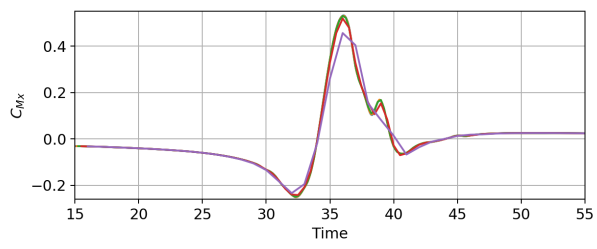
(d) BDF2 Error



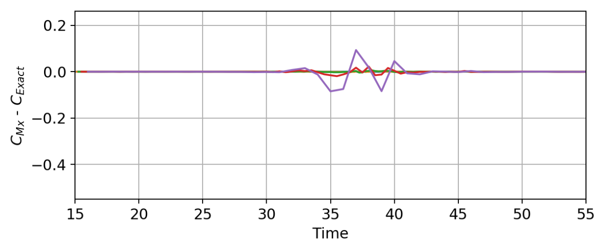
(e) SDIRK22 Value



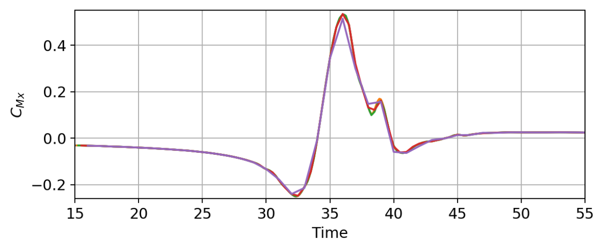
(f) SDIRK22 Error



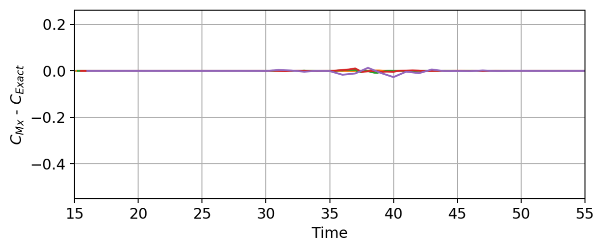
(g) SDIRK33 Value



(h) SDIRK33 Error



(i) SDIRK54 Value



(j) SDIRK54 Error

Figure 5: Pitching moment coefficient over time and error for different timestepping methods. BDF1 is omitted in favor of showing differences between BDF2* and BDF2.

Five timestep sizes were selected from $\Delta t = 0.0625$ to 1.0 and run with BDF1, BDF2, BDF2*, SDIRK22, SDIRK33, and SDIRK54 methods. An "exact" solution is obtained using SDIRK54 with $\Delta t = 0.03125$. The error metric chosen for comparison between methods is the X-moment coefficient from time $T = 30$ to $T = 45$ subtracted from the "exact" solution. Figure 5 summarizes the results for each of the cases, though omits BDF1 since it has no practical use. Similar to results from the previous section, the both BDF2 methods shows significant diffusion and dispersion error even at moderate timestep sizes. The SDIRK22 method shows significant improvement over BDF2 and much less dispersion error at high timesteps. SDIRK33 and SDIRK54 show modest improvement over SDIRK22.

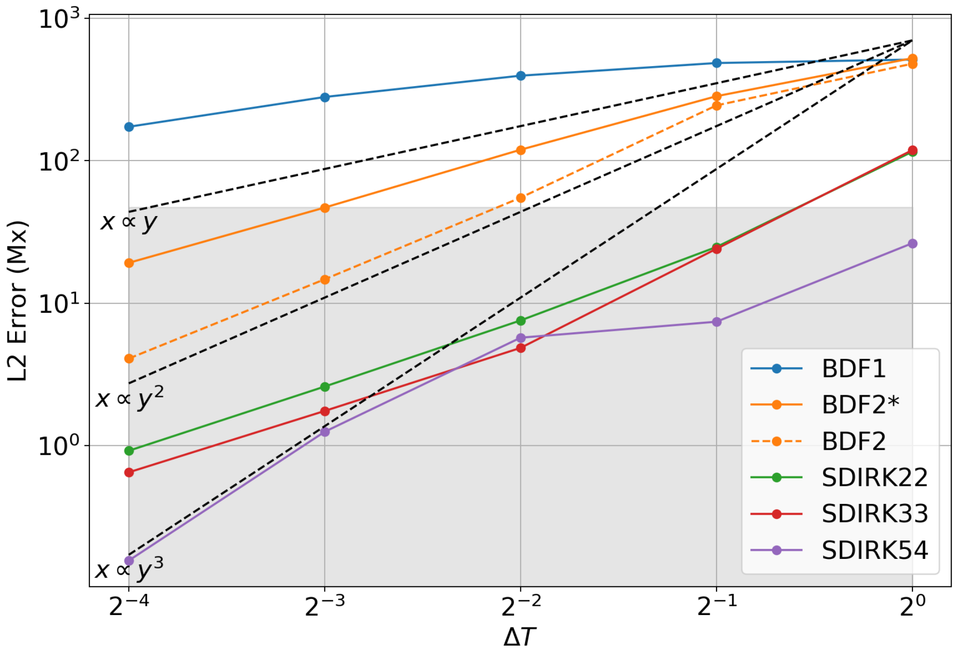


Figure 6: Error Convergence for each method. Points in the shaded region are more accurate than BDF2* at $\Delta t = 2^{-3}$, which is the equivalent step typically used in rotorcraft applications.

Figure 6 shows the L2 norm of the error for each of the timestepping methods. The BDF2* method, which is the baseline Helios operation with less tightly coupled communication, converges at a slope between 1 and 2. When communication is performed every linear iteration the BDF2 method becomes formally second order accurate. The benefits of the formal BDF2 scheme, however, are not enough to enable use of a larger timestep than 2^{-3} while staying in the shaded region.

All SDIRK methods do not match the expected order-slope however the accuracy is still much better compared to BDF schemes. Surprisingly both second and third order methods perform very similarly at all timestep values. The SDIRK54 method appears to converge at a slightly higher rate between 2 and 3. All SDIRK methods up to $\Delta t = 2^{-1}$ stay within the shaded region, meaning in the rotor analogy they should produce improved accuracy over the current operation. With SDIRK22 or SDIRK33 this means a $4\times$ larger timestep can be taken; the SDIRK54 the multiplier jumps to $8\times$.

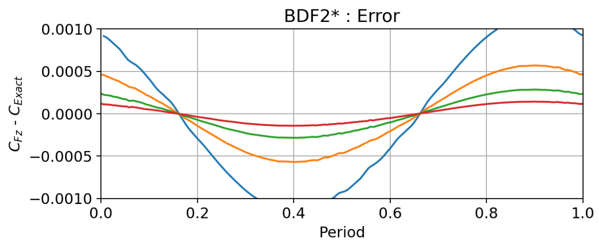
A computational cost comparison cannot be drawn from these results without a few assumptions. Considerations like convergence level, GMRES krylov vectors, and number of non-linear iterations all play a role. As a rough comparison, however, assuming that each stage of a SDIRK method is equivalent in cost to one BDF2 step the SDIRK22, SDIRK33, and SDIRK54 methods would cost approximately $2\times$, $3\times$, and $5\times$ respectively. Therefore the ability to take a $4\times$ larger timestep with SDIRK22 and SDIRK33, or $8\times$ larger timestep with SDIRK54 out-weighs the penalty from the cost of multiple stages.

3.3 Pitching Wing

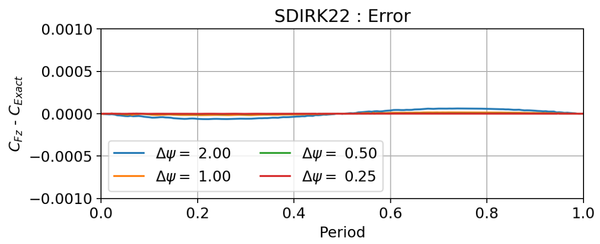
The same grids from Sec. 3.2 are re-used but in a pitching airfoil simulation and without a vortex initialized in the background. The NACA0012 wing pitches about the quarter-chord with a reduced frequency of $k = 0.2$, described in Eqn. 18, and a sinusoidal variation from $\alpha = -5°$ to $\alpha = +5°$. The free-stream Mach number is 0.2, representing predominantly incompressible flow.

$$k = \frac{\Omega c}{2U_\infty}, \quad \alpha = 5sin(\Omega t) \tag{18}$$

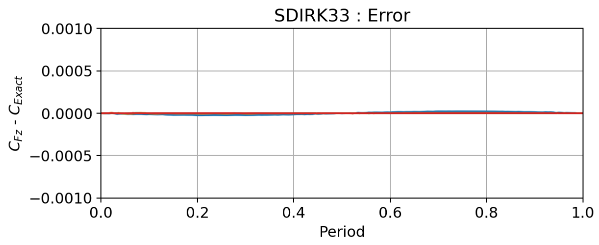
For simplicity the simulation remains inviscid and the forces on the wing over 1 pitching period are compared between different timestepping methods. Since an analytic equation is used for the rigid wing deflection, mStrand uses the exact derivative from the same equation when calculating grid motion terms. Typically a deforming wing or blade would not have a trivial deflection profile and finite differences would instead need to be used in mStrand to find cell face velocities. Four different timesteps are used, equivalent to taking 1440, 720, 360, and 180 steps per period or $\Delta\psi = 0.25°$, $0.5°$, $1.0°$, and $2.0°$.



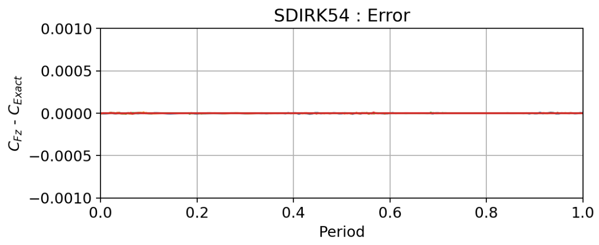
(a) BDF2 Error



(b) SDIRK22 Error



(c) SDIRK33 Error



(d) SDIRK54 Error

Figure 7: Lift coefficient over time and error for each timestepping method.

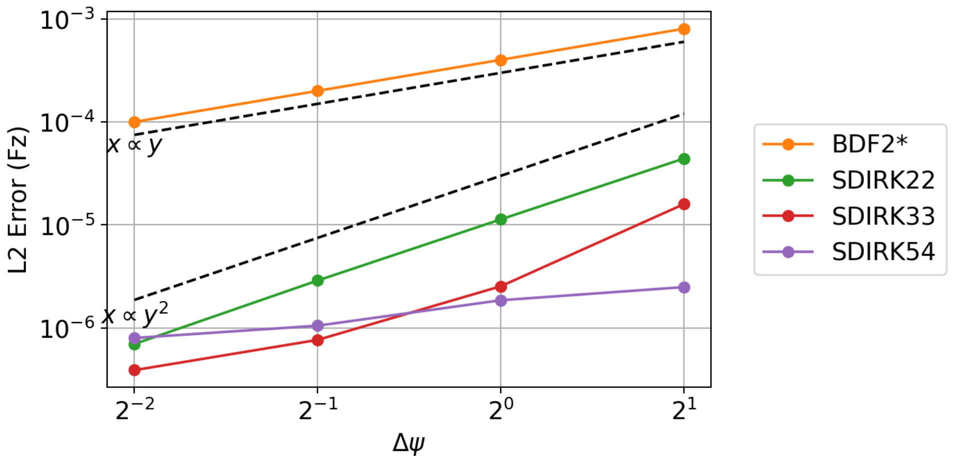


Figure 8: Error Convergence

Figure 7 shows the error in lift for BDF2, SDIRK22, SDIRK33, and SDIRK54 over one oscillation period. The same errors are aggregated into an L2-norm and plotted against timestep in Fig. 8. Similar to results from the wing-vortex case, by moving to SDIRK methods the error can be significantly reduced, even with larger timesteps are used. In fact, with any of the SDIRK methods a timestep of 8× larger can be used compared to BDF2 with the standard $\Delta\psi = 0.25°$ without any accuracy penalty. SDIRK33 and SDIRK54 methods perform similarly for this case, which is a surprise. One possible reason for low convergence slope of SIDIRK33 and SDIRK54 methods is the precision of the data compared is limited to single floating point precision. Perhaps if additional digits were available in the reported engineering quantities the slopes would maintain better agreement with the expected values.

4 Conclusion and Future Work

This work has presented three cases to analyze the performance of high-order temporal methods in an overset framework. The first case is a vortex convection problem in which all methods converge with their expected order of accuracy; the high-order SDIRK methods present promising accuracy and cost benefits over traditional BDF-type schemes. The second case studied is a wing-vortex interaction problem. Though the SDIRK methods did not match up to 4th order temporal convergence, they proved again to be at least more accurate than the BDF methods. The final case involves a pitching wing case in which near- and off-body domains are reconnected each stage of the RK schemes. Again the SDIRK methods show accuracy benefits compared to the traditional BDF2 approach.

Results from this study can be used to extend conclusions about using SDIRK methods for real rotor applications. Vortex convection, blade-vortex interaction, and dynamic stall are all important phenomena in rotorcraft aerodynamics. By studying each of these phenomena independently it is clear that the use of SDIRK methods, even second or third-order accurate, lead to improved accuracy using the same timestep or matched accuracy at larger timesteps compared to traditional approaches.

Future work will see SDIRK methods in Helios applied to hovering rotor calculations to study wake breakdown. Similar to the pitching wing case, a hovering rotor has analytical grid speeds which can be applied in the near-body solver. The rotor case with analytical grid speeds can also be compared to a GCL-consistent finite-difference implementation to validate grid velocity accuracy. Especially use of a high-order, diffusion-dominated scheme like SDIRK33 would be interesting to compare to existing results from Helios.

For each of the three cases in this work the subiterative (non-linear) convergence exceeded eight orders in an effort to isolate temporal error for analysis. An engineering use of high-order methods would more likely converge closer to two orders to save computational resources. A stability and cost study for multi-stage methods converging varying orders is another interesting future research area for this work.

Since each timestep with SDIRK schemes is self-contained and does not use data from previous timesteps, an interesting future use of these schemes would be to investigate adaption in time. For a rotor BVI problem, for example, an engineer may want to use a large timestep for most of the simulation but then reduce to a smaller timestep in the period when BVI is strongest. Alternatively the simulation could be seamlessly switched from second-, to third-, or fourth-order accurate in time for the period of interest.

Acknowledgments

Many of the mStrand-related improvements for this work were developed and debugged with help of Vinod Lakshminarayan. Presented materials are products of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. Funding and support for the presented research is also provided by the U.S. Army Combat Capabilities Development Command, Aviation & Missile Center. Computer resources for some calculations were provided by the DoD HPCMP Frontier Program.

References

[1] Andrew M. Wissink, Dylan Jude, Buvaneswari Jayaraman, Beatrice Roget, Vinod K. Lakshminarayan, Jayanarayanan Sitaraman, Andrew C. Bauer, James R. Forsythe, and Robert D. Trigg. New capabilities in CREATE-AV helios version 11. In *AIAA Scitech 2021 Forum*, Nashville, Tennessee, jan 2021. American Institute of Aeronautics and Astronautics.

[2] Vinod K Lakshminarayan, Jayanarayanan Sitaraman, Beatrice Roget, and Andrew M Wissink. Development and validation of a multi-strand solver for complex aerodynamic flows. *Computers & Fluids*, 147:41–62, 2017.

[3] Dylan Jude, Jayanarayanan Sitaraman, and Andrew Wissink. An octree-based, cartesian navier–stokes solver for modern cluster architectures. *The Journal of Supercomputing*, 78(9):11409–11440, feb 2022.

[4] Pieter G Buning, Dennis C Jespersen, Thomas H Pulliam, WM Chan, Jeffrey P Slotnick, SE Krist, and Kevin J Renze. Overflow user's manual. *NASA Langley Research Center, Hampton, VA*, 2002.

[5] Jennifer Abras and Nathan S. Hariharan. *Investigation of the Impact of Deep Convergence on Numerical Wake Breakdown of Hovering Rotors.*

[6] Zhi Yang and Dimitri J. Mavriplis. Higher-order time integration schemes for aeroelastic applications on unstructured meshes. *AIAA Journal*, 45(1):138–150, 2007.

[7] Li Wang and Dimitri J. Mavriplis. Implicit solution of the unsteady euler equations for high-order accurate discontinuous galerkin discretizations. *Journal of Computational Physics*, 225(2):1994–2015, 2007.

[8] S. Isono and David Zingg. *A Runge-Kutta-Newton-Krylov Algorithm for Fourth-Order Implicit Time Marching Applied to Unsteady Flows.*

[9] Ray Hixon and Miguel Visbal. *Comparison of High-Order Implicit Time Marching Schemes for Unsteady Flow Calculations.*

[10] Jay Sitaraman, Dylan Jude, Beatrice Roget, Steven A. Tran, Jennifer Abras, Jonhoon Bin, and Jeremy Shipman. *Algorithmic Improvements and Capability Enhancements to PUNDIT*, page 1175. 2022.

[11] Beatrice Roget, Jay Sitaraman, and Andrew M. Wissink. *Maneuvering Rotorcraft Simulations Using CREATE A/V<sup>TM</sup> Helios*, page 1057. 2016.

[12] Thomas H. Pulliam and Joseph L. Steger. Implicit finite-difference simulations of three-dimensional compressible flow. *AIAA Journal*, 18(2):159–167, Feb 1980.

[13] C. Kennedy and M. Carpenter. Diagonally implicit runge-kutta methods for ordinary differential equations. a review. In *NASA Technical Report*, Langley, Virginia, 2016. NASA.

[14] Dylan Jude, Jayanarayanan Sitaraman, Vinod Lakshminarayan, and James Baeder. An overset generalised minimal residual method for the multi-solver paradigm. *International Journal of Computational Fluid Dynamics*, 34(1):61–74, jan 2020.

[15] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.