The 9th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2024
3–7 June 2024, Lisboa, Portugal

A HIGH-FIDELITY FRAMEWORK APPROACH ENABLING HIGH-ORDER IMPLICIT TIME STEPPING FOR MDAO

FLORIAN ROSS[1], ADAM BÜCHNER[1], SEBASTIAN GOTTFRIED[1] AND ARTHUR STÜCK[1]

[1] Institute of Software Methods for Product Virtualization
German Aerospace Center (DLR)
Zwickauer Straße 46, 01069 Dresden, Germany
e-mail: florian.ross@dlr.de, arthur.stueck@dlr.de

Key words: Unsteady Multidisciplinary Analysis and Optimization (MDAO), Diagonally Implicit Runge–Kutta (DIRK), MDAO Frameworks, Sensitivity Analysis, Solution Algorithms

Summary. In order to allow high-order time integration of multidisciplinary scenarios via a framework-based approach, we developed RKOpenMDAO. It is an extension to the baseline version of the OpenMDAO framework – which is primarily designed for steady-state like MDAO applications – and enables multidisciplinary time integration via diagonally implicit Runge–Kutta (DIRK) schemes. To yield a robust framework approach capable of time-resolved high-fidelity MDAO, the approach offers strong, nested solution schemes to solve the coupled multidisciplinary systems that arise per time step/stage of the implicit time integration. The implementation is described for the OpenMDAO framework which is used in conjunction with FlowSimulator HPC environment. The verification and demonstration cases presented in the paper benefit from a seamless and modular integration of the CFD solver library CODA behind a unified component/plugin API, which provides access to methods for disciplinary block inversions, residual computations and the calculation of the corresponding derivatives by means of algorithmic differentiation. Numerical experiments are shown for Kaps' problem, a 2D NACA0012 airfoil coupled with a torsion spring, and the ONERA-M6 wing in fully-turbulent flow executed in an MPI-parallel way. We demonstrate that the design order of the high-order Runge–Kutta time integration can be retained for high-fidelity multiphysics problems.

1 INTRODUCTION

Over the last decade, gradient-enabled Multidisciplinary Design Analysis and Optimization (MDAO) frameworks were developed, some of which are available as open-source software such as OpenMDAO [3] or GEMSEO [10]. Features like automated forward- and reverse gradient accumulation over multiple disciplines, nested multidisciplinary solvers and preconditioners, MPI-parallelism, and (open-source) packages of integrated components are attractive to the community in academia and industry to be used in combination with different levels of fidelity in simulation-based analyses. Whereas steady-state MDAO analyses have become state of the art, we see a gap to bridge in both MDAO capabilities and applications for time-resolved problems. A naive framework approach would be consider time-dependent problems as one time-layered coupled problem instantiating the full time-dependency of the model at once in one pseudo-steady system. This would result in the simultaneous allocation of all time steps (and possibly

intermediate data), see e. g. for OpenMDAO [1]. For high-fidelity cases, this leads to prohibitive memory requirements which are avoidable, since most time stepping methods work in a sequential manner and only need data from a low number of previous steps at one point in time.
In this paper, a (missing) MDAO framework ingredient is described to pave the way towards unsteady MDAO: we developed RKOpenMDAO – an open-source extension to OpenMDAO – that allows multidisciplinary time integration on the framework level. Currently, diagonally implicit and explicit Runge–Kutta (DIRK and ERK) are supported. The extension only keeps one time step at a time in memory, storing only the data actually required by the time integration. RKOpenMDAO is set up such that all OpenMDAO functionalities are systematically extended to the unsteady domain, among them the forward/reverse computation of sensitivity derivatives and the monolithic solution capabilities that are required in the context of implicit time integration for large/stiff coupled, nonlinear problems to be solved per Runge–Kutta stage. To this end, it becomes possible to ensure time accuracy with the full construction-order of the Runge–Kutta methods for large-scale, time-accurate MDAO.
In the second section of this paper, the approach and implementation of RKOpenMDAO will be presented in detail. First, an approach to apply DIRK methods monolithically in multiphysics scenarios is explored, and afterwards the implementation details will be discussed. In the third section, numerical experiments conducted with RKOpenMDAO and their results are presented. This includes a simple mathematical example, for which an analytical solution is available, as well as more practical examples using the computational fluid dynamics (CFD) software CODA, developed as part of a collaboration between the French Aerospace Lab ONERA, the German Aerospace Center (DLR), Airbus, and their European research partners.

2 THE FRAMEWORK EXTENSION RKOPENMDAO

2.1 Diagonally implicit Runge–Kutta (DIRK) methods

For the time integration of multidisciplinary problems, we consider a monolithic approach. Instead of each discipline having its own time stepping scheme (possibly with its own step size), one time discretization scheme with one common step size over the full system is applied. There are multiple reasons for this approach, among them: (a) It allows to achieve strong coupling in time in a straight-forward way. Using per-discipline time integration could involve repetition of time steps, interpolation between disciplines due to mismatched integration schemes, and/or more complications we want to avoid; (b) the resulting stage-wise equation systems of this approach are very similar to the ones of the respective stationary problem. Hence, in the transformation from a steady-state MDAO framework approach to an time-accurate one, we can reuse many existing functionalities and leverage synergies; (c) provided that all multidisciplinary components or plugins are HPC-ready and implemented in a scalable way, we can afford to apply the fine time steps – usually dictated by the CFD as the most resource-hungry component – to the other, smaller components without a major loss of overall efficiency. This avoids a number of consistency and conservation issues that would be associated with interpolation in time.

DIRK schemes [5], implemented in the so-called time integrator, were chosen to drive the time integration by the MDAO framework in a monolithic way. These schemes are implicit and offer superior stability properties compared to explicit Runge–Kutta schemes. Moreover, they are easier to implement than fully-implicit Runge–Kutta schemes thanks to the diagonal structure of the Butcher tableau (see Table 1). For a d-dimensional problem with an s-stage

Table 1: General S-stage Butcher tableau of a DIRK scheme.

$$
\begin{array}{c|cccc}
c_1 & a_{11} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & \ddots & 0 \\
c_S & a_{1S} & \cdots & \cdots & a_{SS} \\
\hline
 & b_1 & \cdots & \cdots & b_S
\end{array}
$$

scheme, a series of s similarly structured d-dimensional nonlinear systems has to be solved per time step for DIRK schemes, whereas for fully-implicit ones, a $(s \times d)$-dimensional system has to be solved instead. Nonetheless, there are disadvantages of DIRK methods compared to fully-implicit Runge–Kutta. Due to the diagonal structure of the DIRK schemes, they can at most achieve stage-order two, potentially leading to an order reduction for very stiff problems [6].
For the time integration, one starts with a global multidisciplinary ODE

$$M(t, x(t))x'(t) = f(t, x(t)), \quad x(0) = x_0 \; . \tag{1}$$

Here, $x = (x^{\text{Disc } 1}, x^{\text{Disc } 2}, \ldots, x^{\text{Disc } m})$ represents a multidisciplinary state vector, where the $x^{\text{Disc } i}$ are the (possibly multidimensional) disciplinary states. Applying an ERK or DIRK scheme leads to

$$
\begin{aligned}
x_{n+1} &= x_n + \Delta t \sum_{i=1}^{S} b_i k_i \quad n = 0, \ldots, N-1 \\
M k_i &= f\left(t_n + \Delta t c_i, x_n + \Delta t \sum_{j=1}^{i} a_{ij} k_j\right) \quad i = 1, \ldots, s \; ,
\end{aligned} \tag{2}
$$

where Δt denotes the time step size and t_n is the time at time step n. x_{n+1} and x_n are the global state vectors and k_i is the stage-wise update. Furthermore, a_{ij}, b_i and c_i are coefficients from the Butcher tableau of a Runge–Kutta scheme. The first equation can easily be applied on a framework level per step. However, the second equation cannot directly be represented by an OpenMDAO problem since the number of variables depends on stage of the Runge-Kutta scheme. To resolve this issue, the variable $s_i := \sum_{j=1}^{i-1} a_{ij} k_j$ is introduced:

$$
\begin{aligned}
x_{n+1} &= x_n + \Delta t \sum_{i=1}^{S} b_i k_i \quad n = 0, \ldots, N-1 \\
s_i &= \sum_{j=1}^{i-1} a_{ij} k_j \quad i = 1, \ldots, s \\
M k_i &= f\left(t_n + \Delta t c_i, x_n + \Delta t (s_i + a_{ii} k_i)\right) \quad i = 1, \ldots, s \; .
\end{aligned} \tag{3}
$$

In this form, the equations can be centrally implemented in OpenMDAO as they have the same structure and number of variables regardless of the time step or stage, respectively.

2.2 Implementation

The Equations (3) are implemented in OpenMDAO by nesting two OpenMDAO problems, whereas the top-level problem is responsible the upper two equations. To this end, RKOpenMDAO offers the OpenMDAO component class `RungeKuttaIntegrator`, which encapsulates a loop over all time steps and stages. Framework users need to instantiate such a time integration component in their instationary model and assign a second, inner OpenMDAO problem for the remaining Equation (3) to it. The inner problem must be set up such that for given x_n and s_i, k_i can be calculated by running the model. To extract and insert values of x_n, s_i and k_i, `RungeKuttaIntegrator` identifies the concrete variables in the inner problem from additional user-defined metadata.
Furthermore, the inner problem can make use of nested multidisciplinary solver stacks – a feature offered by (the base-line version of) OpenMDAO for stationary problems. A system-wide nonlinear solver (e.g. a Newton solver) usually embeds multidisciplinary linear solvers (like a GMRES solver), which, in turn, can have multidisciplinary block-preconditioners (like Block-Gauß–Seidel). The latter finally call disciplinary linear solvers which may again forward to their disciplinary linear solver stacks. Time integration is then performed according to Algorithm 1. For a stationary example, the nested solution approach is described in [2] in conjunction with the FlowSimulator HPC ecosystem and CODA, both of which are also used in this work with RKOpenMDAO. The class `RungeKuttaIntegrator` offers calculation of derivatives over time

```
Data: $x_0 = (x_0^{\text{Disc } 1}, x_0^{\text{Disc } 2}, \dots, x_0^{\text{Disc } m})$ global initial state vector
Result: $x_N = (x_N^{\text{Disc } 1}, x_N^{\text{Disc } 2}, \dots, x_N^{\text{Disc } m})$ global state vector after $N$ time steps
State $x \leftarrow x_0$;
Initialize Update_cache: $k \leftarrow [0, \dots, 0]$;
for $i = 1, \dots, N$ do
    Update step metadata in coupled inner problem $(i, t_i, \dots)$;
    Set $x$ into coupled inner problem;
    for $j = 1, \dots s$ do
        Update stage metadata in coupled inner problem $(j, t_i^j, \dots)$;
        if $j = 1$ then
            Clear $k$;
            $s_i \leftarrow 0$;
        else
            $s_i \leftarrow \sum_{j=1}^{i-1} a_{ij} k_j$;
        end
        Set $s_i$ into coupled inner problem;
        Solve coupled inner problem;
        Copy $k_i$ from coupled inner problem;
    end
    $x \leftarrow x + \Delta t \sum_{i=1}^{s} b_i k_i$;
end
$x_N \leftarrow x$
```

Algorithm 1: Pseudocode for the time integration of the `RungeKuttaIntegrator` with given s-stage Butcher tableau (A, b, c) and step size Δt.

both in forward and reverse mode via the usual interface of OpenMDAO. In forward mode, directional derivatives are computed on-the-fly during the time integration. In reverse mode, the multidisciplinary time trajectory needs to be traversed backwards. (Re)storing the complete multidisciplinary states to (from) disk in every intermediate time stage/step would lead to excessive memory requirements. Thus, memory can be traded for CPU-time by checkpointing and recomputation via PyRevolve. Finally, RKOpenMDAO is fully compatible with MPI-distributed problems. Provided the inner problem is parallelized via the parallelization features of OpenMDAO, namely distributed variables or parallel groups, the `RungeKuttaIntegrator` picks up the distribution of data and organizes its own internal data in the same pattern.

3 NUMERICAL EXPERIMENTS

3.1 Kaps' Problem

The first problem we consider is Kaps' problem [8, 5], a singularly disturbed problem described by the following equations

$$\begin{aligned} \epsilon y_1'(t) &= -(1+2\epsilon)y_1(t) + y_2^2(t)\ , \\ y_2'(t) &= y_1(t) - y_2(t) - y_2^2(t)\ , \\ y_1(0) &= y_2(0) = 1\ . \end{aligned} \tag{4}$$

By varying the parameter ϵ, the stiffness of the equations can be controlled. With the choice

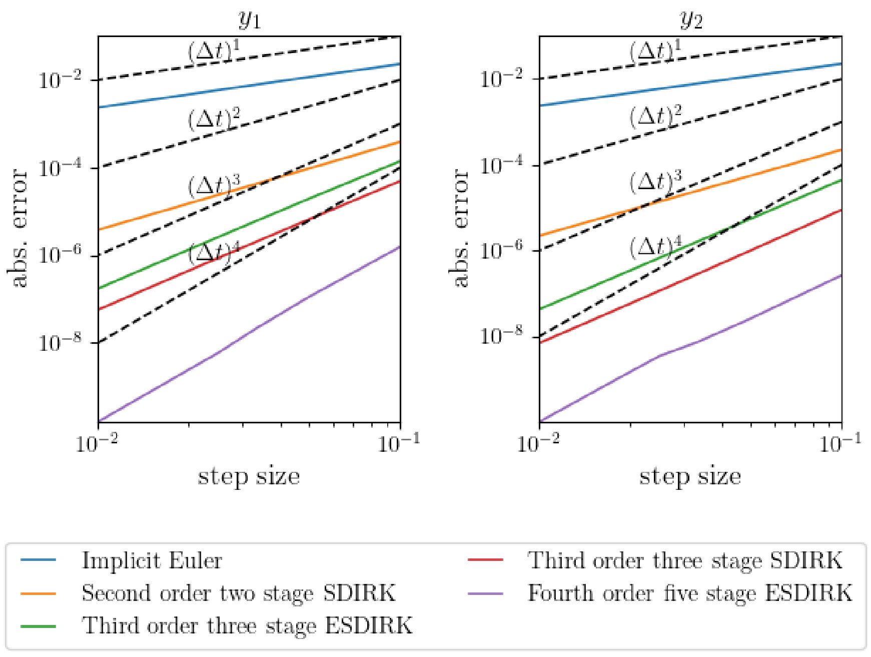

Figure 1: Kaps' problem with $\epsilon = 1.0$, a non-stiff case. The theoritical order of all tested time integration scheme is retained.

of $\epsilon = 0$, the problem becomes a differential-algebraic equation (DAE). The problem is used

to verify that RKOpenMDAO reproduces the design order of different DIRK schemes and parameter choices found in [5]. In particular, the implicit-Euler method, a second-order two-stage SDIRK method [5], a third-order three-stage ESDIRK [5], a third-order three-stage SDIRK [5] and a fourth-order five-stage ESDIRK [5] were used. We used the Butcher tableaux and optimal parameters described in [5], p. 72–86. SDIRK (singly-diagonal implicit Runge–Kutta) methods have the property that all their diagonal entries of the matrix A in their Butcher tableau are the same. The same is true for ESDIRK (explicit singly-diagonal implicit Runge–Kutta) methods, with the exception of the first diagonal entry, which is 0.
The solution of these equations is always the same, independent of the choice of of ϵ

$$y_1(t) = \exp(-2t) \ ,$$
$$y_2(t) = \exp(-t) \ .$$

In all following cases, step sizes were varied between 10^{-1} and 10^-2, with the end time of $t = 1$. The original order of the DIRK methods is retained as expected for non-stiff cases ($\epsilon = 1$) as illustrated in Figure 1. For more stiff cases, e. g. $\epsilon = 0.001$, an order reduction can be observed

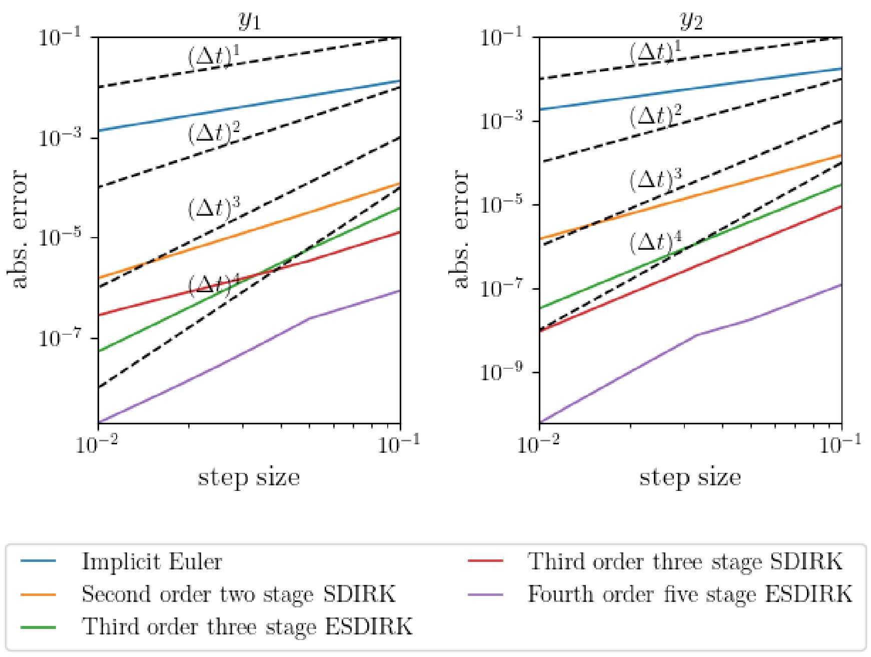

Figure 2: Kaps' problem with $\epsilon = 0.001$, a stiff case. Here there is observable order reduction in the first variable when using the third-order SDIRK scheme and the fourth-order ESDIRK scheme. The order reduces expectedly to 2 respectively 3, since this is the stage order of the method plus one.

for the first variable as shown in Figure 2. Both the order of the third-order SDIRK method and the fourth-order ESDIRK method is reduced by one. This result is expected, as in stiff cases, the order of DIRK schemes may be reduced to stage-order plus one [5, p. 16]. In the case of SDIRK methods the order reduces to 2, since the stage order is limited to 1. For ESDIRK methods, stage order can also be 2, as is the case with the two methods used here. For the DAE

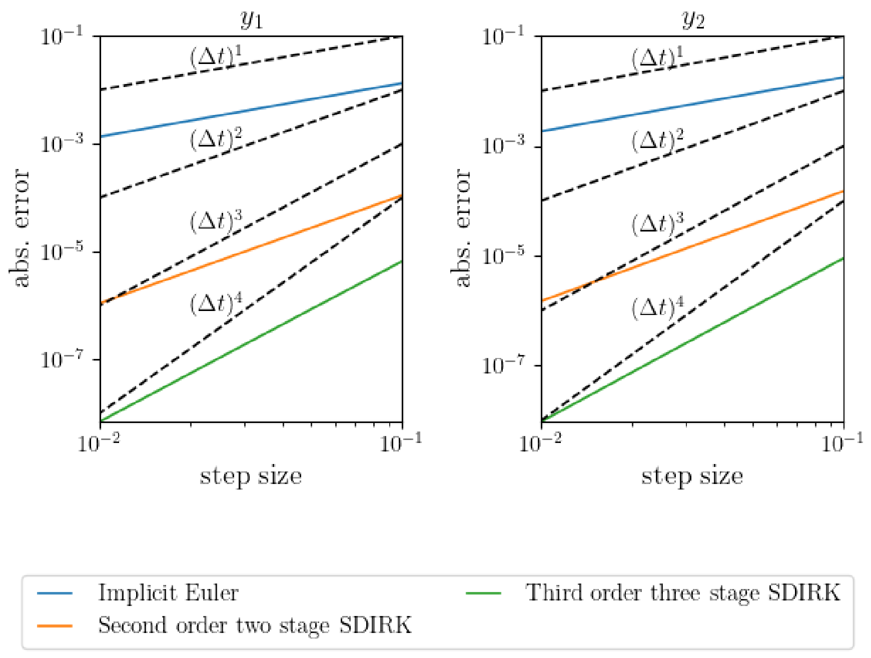

Figure 3: Kaps' problem with $\epsilon = 0.0$, a DAE case. There is no order reduction for the tested SDIRK schemes. This is to be expected, since the problem is only a DAE of index 1.

case ($\epsilon = 0$), no results for ESDIRK schemes are provided, since the current implementation does not support DAEs while using methods with explicit stages. For the SDIRK schemes, the implementation still achieves the theoretical order of the tested methods (see Figure 3).

3.2 Multiphysics MDA Problem

We considered the flow around a NACA0012 airfoil coupled to a torsional spring to investigate the effect of different coupling methods on the order of the Runge–Kutta time integration. The coupling works in the following way: a torsional spring is attached at the quarter chord of the airfoil. The pitching moment of the spring acts on the airfoil together with the integral aerodynamic moment. Starting from an initial angle of attack, it results in an oscillating pitching movement of the airfoil over time.
After spatial discretization of the Euler equations via the finite volume method, the following ODE system is obtained

$$\begin{aligned} 0 &= \frac{\partial}{\partial t} W + R(W, \alpha) \ , \\ 0 &= I \frac{\partial^2}{\partial t^2} \alpha + \mu \alpha - M_y(W) \ , \end{aligned}$$

wherein W denotes the flow state consisting of the conservative variables for mass, momentum and energy, and $M_y(W)$ is the aerodynamic pitching moment. Transforming the second-order ODE part to first order and applying a DIRK scheme leads to the following system that needs to be solved per Runge–Kutta stage

$$
\begin{aligned}
0 &= \frac{W_n^i - W_n - \Delta t s_i^W}{\Delta t a_{ii}} + R(W_n^i, \alpha_n^i) \\
0 &= \frac{W_n^i - W_n - \Delta t s_i^W}{\Delta t a_{ii}} - k_i^W \\
0 &= \frac{\alpha_n^i - \alpha_n - \Delta t s_i^\alpha}{\Delta t a_{ii}} - \dot{\alpha}_n^i \\
0 &= \frac{\alpha_n^i - \alpha_n - \Delta t s_i^\alpha}{\Delta t a_{ii}} - k_i^\alpha \\
0 &= I \frac{\dot{\alpha}_n^i - \dot{\alpha}_n - \Delta t s_i^{\dot{\alpha}}}{\Delta t a_{ii}} + \mu \alpha_n^i - M_y(W_n^i) \\
0 &= \frac{\dot{\alpha}_n^i - \dot{\alpha}_n - \Delta t s_i^{\dot{\alpha}}}{\Delta t a_{ii}} - k_i^{\dot{\alpha}} \ .
\end{aligned}
$$

The above equations are implemented into an OpenMDAO problem via components for CFD

Table 2: Reference values for nondimensionalization.

Chord length	1 m	Farfield Temperature	300 K
Farfield Pressure	10^5 kg m^{-1} s^{-2}	Farfield Density	1.3 kg m^{-3}

solver software CODA [4] and for the spring. The CFD component is implemented using the API methods of the CFD library CODA, while the formulas for the spring component are directly implemented in a dedicated implicit component as written above. In conjunction with the quantities listed in Table 2, we used a Mach number of $Ma = 0.1$ together with the nondimensional values $I = 0.7692$, and $\mu = 10^{-5}$. For the spring, we used the initial values $\alpha_0 = 10°$ and $\dot{\alpha}_0 = 0$. W_0 was set to the steady-state solution at the same angle of attack. The dimensional reference time 0.0036 s is obtained according to the choices listed in Table 2.
For the following numerical experiments, a reference solution to the problem was generated from the classical explicit fourth-order Runge–Kutta method with the fine nondimensional time step $\Delta t = 10^{-4}$. The unsteady simulation ran until the nondimensional time $T = 1$ was reached. The unstructured computational mesh consisted of around 2000 finite volumes in parallel execution with 8 MPI processes. This reference solution was then compared to similar runs with the same DIRK schemes introduced in the last example. Step sizes from 10^{-1} to 10^{-2} were used to asses the resulting order of the schemes in a method-of-lines approach – i.e. the spatial mesh was kept constant while we modified the physical time step. For all cases, the solution per stage was computed using a system-wide Newton-Krylov-solver with multidisciplinary Block-Jacobi preconditioning. As seen in in Figure 4, the first- and second-order methods replicate the design order almost perfectly. For higher-order methods, an order-reduction was observed for larger step sizes, but as the step size progresses towards 10^{-2}, the theoretical design order is obtained.

These results were gained via a strong coupling approach, i.e. data between the airfoil and spring is frequently exchanged in every iteration of the nested iteration schemes on the level of

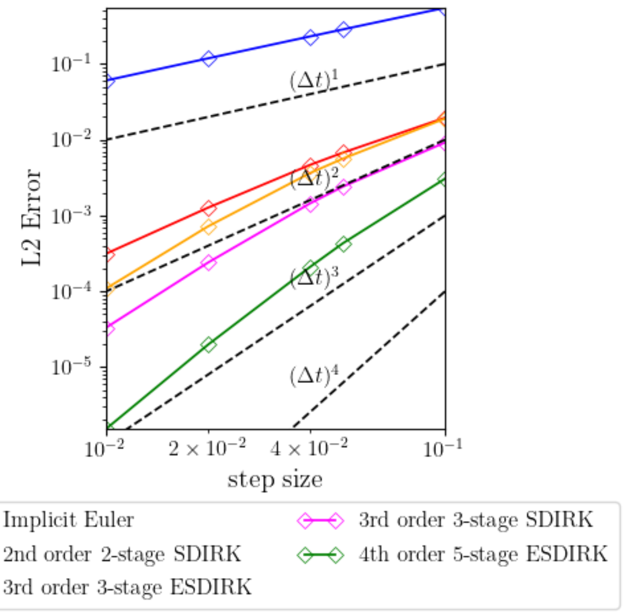

Figure 4: Error between the base solution and the DIRK solutions in the NACA0012-spring-coupling problem. All methods can reach their design order if an sufficiently small step size is used.

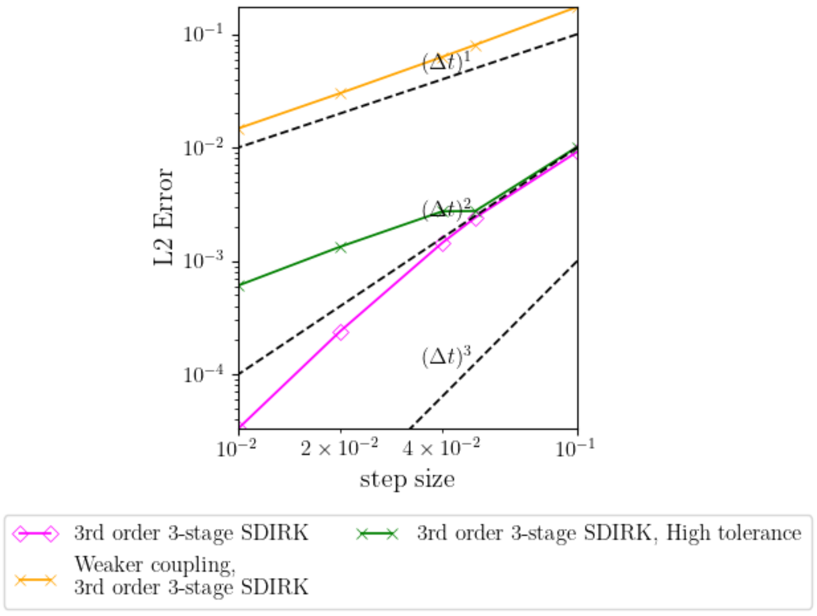

Figure 5: Error between the base solution and the DIRK solution of a third order DIRK scheme with both a stronger and weaker coupling approach for the NACA0012-spring-coupling problem, as well as the error to the modified strong coupling problem with relaxed solver tolerances. The strongly-coupled solution yields the expected order as the step size are decreases. Meanwhile, the weakly-coupled solution only achieves order 1. The solution with looser solver tolerances follows the graph of the stricter tolerance down to a certain step size. From that point on, it only achieves order 1.

Runge–Kutta stages. In the following, the solver coupling on the stage-level is relaxed to artificially weaken the solver and analyse its effect on the time integration. In this weaker coupling, the spring still gets the new moment from the CFD in each time stage, but the CFD will only use the angle α_n from the beginning of the time step. Figure 5 shows that the weak coupling falls back to order 1, compared to the theoretical design order 3. This highlights the benefit of strong coupling in order to retain the higher-order and the accuracy of the time integration. Furthermore, Figure 5 shows the consequences implied by using a weaker solver tolerance for the nonlinear solver, resulting in a higher iteration error per Runge–Kutta stage. The iteration was stopped at a relative tolerance of 10^{-2}, while the original strong coupling solution was converged until machine accuracy was reached. The behaviour is the same as with the strict tolerances until a certain step size is reached. This indicates that it is necessary to converge the stage-wise problem sufficiently accurate in order to reach the desired higher-order convergence in time for the multidisciplinary time integration. Though it is interesting to note that even after that point, convergence to the time step independent solution still continued with order 1 instead of breaking down.

3.3 Instationary Hi-Fi CFD

The last example here is the flow around an ONERA M6 wing [7]. This is a single-disciplinary test case, based on which we want to investigate the scaling behaviour of the framework-driven time integration and compare it against the performance-optimized time integration implemented in the CFD solver CODA [4]. Note that this exercise is only carried out to quantify the overhead associated with the framework-driven time integration and does not yield any benefits from a functionality point-of-view in the single-disciplinary case. The simulation includes a jump of the angle of attack from 1° to 2° at the start of the simulation. It uses the compressible RANS equations with negative extension of the Spalart–Allmaras turbulence model (SA-neg), with $Ma = 0.84$ and $Re = 14.6 \cdot 10^6$. The unsteady simulation was ran with 50 time steps with a nondimensional step size $\Delta t = 0.01$. The unstructured computational mesh consisted of around $2.2 \cdot 10^5$ finite volumes. For time integration, a third-order ESDIRK scheme that is equivalently available in the CFD-solver CODA was used. These settings were applied both for CODA standalone and for the framework-driven time integration by means of RKOpenMDAO. It was run on varying numbers of cores to compare the runtimes of the internal time integration of CODA against the framework-driven approach. Additionally, the simulation results were compared, and found to only differ by machine accuracy. The runtimes obained are presented in Figure 6. The runtime of the framework-driven approach shows a systematic overhead relative to CODA-standalone execution. The scaling behaviour is similar for lower core counts. This is to be expected, since the framework-driven approach induced additional overhead, as there are many , process-local copy operations of data between CODA, the inner problem of the `RungeKuttaIntegrator`, and the integrator itself. However, with larger core counts, a decrease of the scaling in the framework approach compared to CODA can be observed. This behaviour is unexpected and further studies are required to understand its origin. Systematic profiling of the framework workflow (beyond RKOpenMDAO) will be carried out to pinpoint the mechanism. For example, different code paths taken by the two approaches could explain the differences in the observed scaling behaviour.

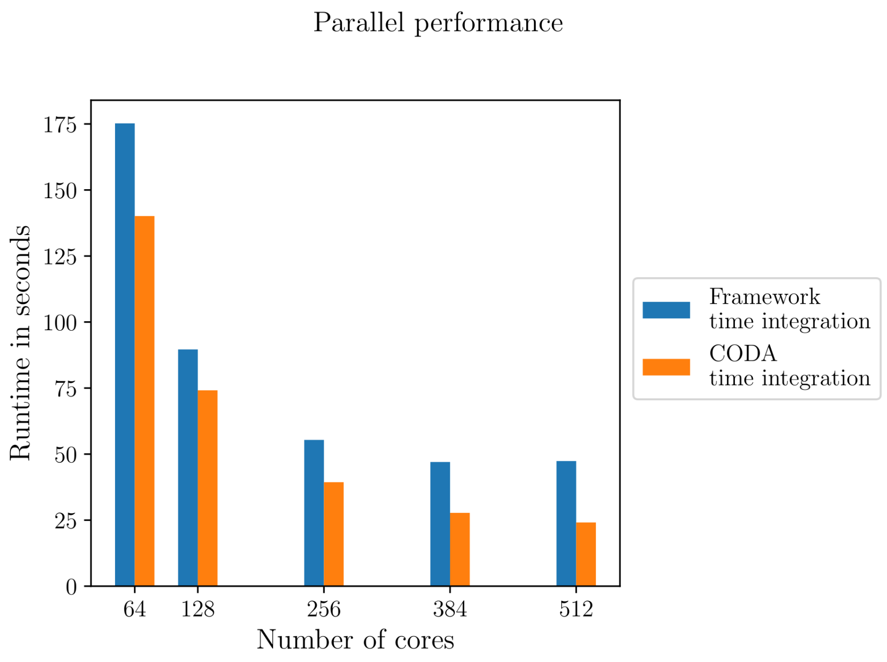

Figure 6: Comparison of the runtime over core counts obtained from CFD-standalone time integration and the framework-driven time integration by RKOpenMDAO. It is observed that the latter introduces an overhead, e.g. due to data transfer between CODA and RKOpenMDAO.

4 CONCLUSIONS

We developed an extension to OpenMDAO that allows the time integration of instationary high-fidelity problems. This extension allows strong coupling of multiple disciplines in time to enable highly-accurate, high-order time integration in MDAO. The approach was verified and the framework capabilities were demonstrated for academic analytic problems and a CFD case of different problem sizes.
In future work, further time integration schemes are to be implemented in the multidisciplinary time integrator, for example in the form of General Linear Methods (GLM) [9, Ch. 5]. This in turn would grant the ability to integrate with Runge–Kutta, linear multistep, and mixed methods using one common implementation. Moreover, we want to use the high-order time-integration to carry over adaptive time stepping, which is well known on the disciplinary level, to MDAO problems.
As shown in the previous paragraph we are going to investigate the scalability of the framework-driven time integration, also in combination with larger high-fidelity MDAO cases like aeroelastic aircraft configurations. Along these lines, the spring will be replaced by a state-of-the-art finite-element CSM solver. Finally, we want to use RKOpenMDAO for unsteady optimization problems, in which it is used to integrate coupled adjoint problems backwards in time with checkpointing.

Acknowledgments The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARA is partially funded by ”Saxon State Ministry for Economic Affairs, Labour and Transport” and ”Federal Ministry for Economic Affairs and Climate Action”.

REFERENCES

[1] Hwang, J.T., and Munster, D.W. 2018. Solution of differential equations in gradient-based multidisciplinary design optimization. 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. Kissimmee.

[2] Masilamani, K., Büchner, A., Ehrmanntraut, S., Gottfried, S., Reimer, L., and Stueck, A. 2024. A Scalable MDAO Framework Approach Enabling Nested Solution Algorithms for Static Aeroelastic Coupling Scenarios. AIAA AVIATION FORUM AND ASCEND 2024 (p. 4403).

[3] Gray, J.S, Hwang, J.T, Martins, J.R.R.A., Moore, K.T. and Naylor, B.A. 2019. ”OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization”. In Structural and Multidisciplinary Optimization, vol. 59: 1075–1104. http://doi.org/10.1007/s00158-019-02211-z

[4] Leicht, T. et al. 2016. DLR-Project DIGITAL-X – Next Generation CFD Solver FLUCS. DLRK 2016

[5] Kennedy, C. and Carpenter, M. 2016. Diagonally Implicit Runge-Kutta Methods for Ordinary Differential Equations. A Review.

[6] Hairer, E., Lubich, C. and Roche, M. 1988. Error of Runge-Kutta methods for stiff problems studied via differential algebraic equations. In BIT Numerical Mathematics 28, 678–700. https://doi.org/10.1007/BF01941143

[7] Rumsey, C. 3D ONERA M6 Wing Validation Case [online]. 2021-11-10. [visited on 2024-07-30]. NASA, Available from: https://turbmodels.larc.nasa.gov/onerawingnumerics_val.html

[8] Dekker, K. and Verwer, J.G. 1984. Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations, North-Holland, Amsterdam, Netherlands

[9] Butcher, J.C. 2016. Numerical Methods for Ordinary Differential Equations. John Wiley & Sons, Incorporated. https://doi.org/10.1002/9781119121534

[10] Gallard, F., Vanaret, C., Guénot, D, et al. 2018. GEMS: A Python Library for Automation of Multidisciplinary Design Optimization Process Generation. In : 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. p. 0657.