2023 European Control Conference (ECC)
June 13-16, 2023. Bucharest, Romania

Fast integrators with sensitivity propagation for use in CasADi

Jonathan Frey[1,2], Jochem De Schutter[1] and Moritz Diehl[1,2]

Abstract— **Efficient integrators with sensitivity propagation are an essential ingredient for the numerical solution of optimal control problems. This paper gives an overview on the `acados` integrators, their `Python` interface and presents a workflow that allows using them with their sensitivities within a nonlinear programming (NLP) solver interfaced by `CasADi`. The implementation is discussed, demonstrated and provided as open-source software. The computation times of the proposed integrator and its sensitivity computation are compared to the native `CasADi` collocation integrator, `CVODES` and `IDAS` on different examples. A speedup of one order of magnitude for simulation and of up to three orders of magnitude for the forward sensitivity propagation is shown for an airborne wind energy system model.**

I. Introduction

Numerical optimization has become more widely used and tractable for increasingly complex problems due to advances in software and hardware. When optimizing design and configurations of physical systems, one usually deals with optimization problems that are constrained by the evolution of a dynamic system model. In the context of offline optimization, such models are often complex, stiff and associated with a high computational burden. However, such accurate models can improve optimality compared to solutions obtained by less accurate models.

The software package `CasADi` [1] is a widely used open-source framework for algorithmic differentiation and optimization which comes with interfaces to a variety of solvers for nonlinear programming (NLP) and initial value problems (IVP). The open-source interior-point NLP solver `IPOPT` [2] is probably the most widely used solver within `CasADi` and known for its numerical robustness. However, an optimization solver derived from `CasADi` and `IPOPT` is typically not suitable for application in fast embedded optimal control applications. In this context, more specialized software, like `acados` [3], which implements SQP-type algorithms and integration methods that can efficiently propagate first and second order sensitivities, often needs to be used. For efficiency, `acados` relies on the basic linear algebra package `BLASFEO` [4] and the (partial) condensing functionality implemented in `HPIPM` [5].

In this paper, we present a novel open-source software that allows one to use the `acados` integrators with their efficient native sensitivity propagation within a `CasADi` NLP obtained from a direct multiple shooting [6] parametrization. The use of `acados` integrators within `IPOPT` allows solving NLPs which include the simulation of complex dynamical systems with one order of magnitude faster computation times compared to native collocation-based integrators provided by the `CasADi` software package. Additionally, this workflow enables users to have a smoother transition from a `CasADi` + `IPOPT` implementation of their problem specific solver, to an `acados` implementation by using the intermediate step of using the `acados` integrator inside `IPOPT`.

The paper is organized as follows: Section II gives a quick overview on integrators and sensitivity propagation for use within optimization. Section III gives an overview on the `acados` integrators, their interfaces to `Python`, and the `CasADi` wrapper. Section IV shows some numerical experiments using the `acados` integrators within `CasADi`.

II. Theoretical Background

This section gives a brief overview on integrators for use within numerical optimization algorithms, [6].

A. Integrators for DAE and sensitivities

Nonlinear dynamic systems are often given in form of a differential algebraic equation (DAE),

$$f^{\mathrm{impl}}(\dot{x}(t), x(t), u(t), z(t)) = 0, \tag{1}$$

where the function $f^{\mathrm{impl}} : \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{x}} \times \mathbb{R}^{n_\mathrm{u}} \times \mathbb{R}^{n_\mathrm{z}} \rightarrow \mathbb{R}^{n_\mathrm{x}+n_\mathrm{z}}$ describes the evolution of the dynamic system consisting of state x, control input u and an algebraic state z over time t. We assume that the DAE (1) is of index-1, i.e. the matrix $\frac{\partial f^{\mathrm{impl}}}{\partial(\dot{x},z)}(\cdot)$ is invertible and note that higher index DAEs are often reformulated as index-1 using index-reduction techniques.

We refer to an algorithm that solves the initial value problem (IVP) of DAE (1) together with an initial state $x_0 = x(0)$ and a given constant control input u_0 for a simulation time T_{sim} as an *integrator*

$$\Phi(x_0, u_0) \approx x(T_{\mathrm{sim}}). \tag{2}$$

Additionally, when solving optimization problems constrained by nonlinear dynamic systems, the integrator should be able to provide the forward sensitivities of the result with respect to the initial state and control input, i.e.

$$\frac{\mathrm{d}\Phi(x_0, u_0)}{\mathrm{d}(x_0, u_0)} \in \mathbb{R}^{n_\mathrm{x} \times (n_\mathrm{x}+n_\mathrm{u})}. \tag{3}$$

On the other hand, computing adjoint sensitivities directly is more efficient within some optimization algorithms, such as adjoint-based inexact SQP methods, or when BFGS hessian

[1]Department of Microsystems Engineering (IMTEK), University Freiburg, 79110 Freiburg, Germany {name.surname}@imtek.uni-freiburg.de
[2]Department of Mathematics, University Freiburg, Germany

approximations are used. The adjoint sensitivities for a given seed vector $\lambda \in \mathbb{R}^{n_\mathrm{x}}$ are defined as

$$\frac{\mathrm{d}\lambda^\top \Phi(x_0, u_0)}{\mathrm{d}(x_0, u_0)} \in \mathbb{R}^{1\times(n_\mathrm{x}+n_\mathrm{u})}. \tag{4}$$

Moreover, second-order sensitivities are needed when using any optimization method that relies on an exact Hessian of the Lagrangian. The second order sensitivities for a given seed vector $\lambda \in \mathbb{R}^{n_\mathrm{x}}$ are given as

$$\frac{\mathrm{d}^2\lambda^\top \Phi(x_0, u_0)}{\mathrm{d}^2(x_0, u_0)} \in \mathbb{R}^{(n_\mathrm{x}+n_\mathrm{u})\times(n_\mathrm{x}+n_\mathrm{u})}. \tag{5}$$

B. Runge-Kutta methods in `acados`

A Runge-Kutta method is defined by the equations

$$0 = f^\mathrm{impl}(k_i, x_0 + T_\mathrm{sim}\sum_{j=1}^{s} a_{ij}k_j, u_0, z_j) \tag{6a}$$

$$x_+ = x_0 + T_\mathrm{sim}\sum_{j=1}^{s} b_j k_j. \tag{6b}$$

where the values a_{ij}, b_i, c_i for $i, j = 1, \dots, s$ form the Butcher tableau. In general, (6a) defines a set of implicit equations which typically are solved by applying a fixed number of Newton iterations. Subsequently, the output is computed via (6b).

In case of an explicit ODE, of the form

$$\dot{x}(t) = f^\mathrm{expl}(x(t), u(t)), \tag{7}$$

equation (6a) simplifies to a set of explicit equations, if the matrix A only has values below its diagonal. Such methods are called explicit Runge-Kutta methods (ERK).

The more generally applicable implicit Runge-Kutta (IRK) methods have better stability properties and a higher order of integration. The `acados` IRK method supports the Gauss-Legendre methods, which are of order $2s$ and A-stable, and the Gauss-Radau IIA methods, which are of order $2s-1$ and L-stable, which is desirable when handling stiff systems, [6]. Additionally, it is common to use an equidistant partition of the simulation interval and apply the Runge-Kutta method n_steps times.

Within `acados`, ERK and IRK integrators with efficient sensitivity propagation (forward, adjoint and second-order) are implemented. The `acados` IRK method implements the efficient symmetric Hessian propagation technique presented in [7]. Interfaces to conveniently create integrators exist for `Python` and MATLAB, which use the algorithmic differentiation (AD) and code generation functionality of `CasADi` to set up the ODE or DAE function and their derivatives.

When applying an SQP-type Optimal Control Problem (OCP) solver in an embedded context, it is typical to use an IRK method with a fixed number of Newton iterations [8] to reduce the variance of the OCP solution time towards a deterministic runtime. However, outside of this context, it makes sense to (additionally) use a tolerance up to which the system of integration equations needs to be solved as a termination criterion, which has been added to the `acados` IRK implementation.

Additionally, the GNSF-IRK method [9] is implemented in `acados`, which can exploit different kinds of linear dependencies within a DAE and solve a nonlinear system in a reduced space instead of (6a). However, second-order sensitivity propagation is not implemented in this integrator yet, limiting its use to optimization methods without exact Hessians.

C. Direct Multiple Shooting and Direct Collocation

The two most general direct approaches for solving continuous time OCPs are (direct) multiple shooting and direct collocation, [6]. Within multiple shooting, the dynamic system is discretized on each shooting interval using an integrator, which handles the integration variables (k_i in the case of a RK method) internally. On the other hand, a *direct collocation* discretization directly handles the discretization within the optimization, by using the integration variables as optimization variables and the integration equations, (6a) in case of a RK method, as constraints of the NLP. The direct collocation NLP has significantly more optimization variables and a more sparse structure.

While there exist a variety of solvers that exploit the sparsity pattern of a multiple shooting based NLP or QP, solving the direct collocation based NLP is more attractive when using a solver that can exploit general sparsity patterns, such as `IPOPT`. Moreover, interior-point NLP solvers can handle nonlinear dynamical systems via direct collocation more robustly, since the globalization strategies are applied on the full problem, including integration equations. On the other hand, the multiple shooting based solution can often lead to a reduced memory footprint. Additionally, it is essential for convergence on the multiple shooting based NLP that the sensitivity propagation of the integrator is consistent with its nominal evaluation, which can be achieved by freezing all adaptive settings, such as the internal step sizes, order, and iteration matrices. The performance of the two discretization techniques highly depends on the dynamic system of interest, the required integration order and the available integrators and NLP solvers, which makes a general comparison very hard.

Note that in [10] the concept of *lifted integrators* was introduced which enables one to apply an SQP-type method with the classic OCP structured sparsity pattern, while iterating equivalently as on a direct collocation based problem.

III. Implementation

This section gives an overview of existing integrators in `CasADi`, implementation details of the `acados` integrators that make them fast, and their `Python` interface. Finally, the implementation of making the `acados` integrators fully usable in a `CasADi` NLP description and solver is discussed. This feature has been released as open-source software [11].

A. State-of-the-art in `CasADi`

The `CasADi Integrator` class supports the following implementations:

- explicit RK-4 method

- Collocation (IRK) method
- CVODES [12]: ODE solver from the Sundials suite [13]
- IDAS: DAE solver from the Sundials suite [13]

The CasADi collocation integrator can be used with Gauss-Legendre or Gauss-Radau IIA Butcher tableaux and internally uses the Rootfinder class. In contrast to the IRK implementation in acados, it does not use the integration variables k_i from equation (6a), but instead the quantities $x_0 + T_{\mathrm{sim}} \sum_{j=1}^{s} a_{ij} k_j$.

CVODES implements linear multistep methods such as Adams-Moulton (AM) methods and Backward differentiation formulas (BDF) with adaptive step-sizes. IDAS mainly implements BDF formulas. Both CVODES and IDAS offer forward and adjoint sensitivity analysis.

The CasADi integrator interface allows one to specify quadrature states, which are quantities that do not influence the behavior of the dynamic system, but only depend on it and can thus be simulated more efficiently. Note that this kind of quadrature states are a special case of a linear output system used in [14] and the GNSF structure [9].

Note that none of the baseline CasADi integrators can be code-generated, which could yield a speedup in run-time.

B. *acados integrators*

The model functions and their derivatives are code-generated by CasADi, then compiled and linked with the acados library, in the standard workflow of the high-level acados interfaces. Thus, the *creation time* of an acados integrator is significantly longer compared to the CasADi integrators mentioned above. However, when using such an integrator inside an optimization solver, especially, when solving multiple problems, the reduced *run-time* quickly makes up for the longer creation time.

All linear algebra operations in the implicit acados integrators are performed using BLASFEO. The forward sensitivity propagation is performed efficiently using the implicit function theorem, following [8]. The implicit acados integrators leave the integration variables from the last call in the memory, which has found to be useful in the context of MPC and leads to less Newton iterations in the numerical experiments in this paper. It is possible to reset the initial guesses of the integration variables via the interfaces, which could be done in the proposed CasADi wrapper.

C. *acados integrator interface*

The acados interface to Python offers convenient methods to create an acados integrator, i.e., an instance of AcadosSimSolver, and to interact with it. Possible interactions with AcadosSimSolver include setting the initial state, the control input, the parameter values, the guess (initialization) of the integration variables, the time horizon and the adjoint seed. Most options, such as the size of the Butcher tableau s, or the number of integration steps n_{steps} cannot be changed dynamically, since the memory size of the integrator depends on them and memory allocation should only happen once to create the integrator. However, we added the possibility to set options for the sensitivity propagation modes, if the initial memory allocation took them into account.

The default Python wrapper of the acados solvers uses ctypes. In order to speed up interactions with the Python object, we added a cython wrapper to the AcadosSimSolver. This wrapper translates the code for the interaction layer into C code and compiles it. Such a wrapper is also available for the acados OCP solver and in use in the open source driving assistance system openpilot [15].

$$\begin{tikzcd}
\texttt{CasadosIntegrator} \arrow[d, "\texttt{get\_jacobian()}"] \arrow[dd, bend left, "\texttt{get\_reverse()}"] \\
\texttt{CasadosIntegratorSensForw} \\
\texttt{CasadosIntegratorSensAdj} \arrow[d, "\texttt{get\_jacobian()}"] \\
\texttt{CasadosIntegratorSensHess}
\end{tikzcd}$$

Fig. 1. Derivative implementation using the CasADi Callback.

D. *CasADi wrapper for acados integrators*

We use the CasADi Callback class to wrap the interactions with the acados integrator into a CasADi Python object, called CasadosIntegrator. The CasADi Callback class offers a public API to the FunctionInternal class. We provide the implementation of the virtual methods that the Callback class needs to provide in order to be functional within the CasADi workflow. Particularly, we implement forward and adjoint derivative computation via the functions has_jacobian() and get_jacobian(), respectively has_reverse() and get_reverse(), as illustrated in Figure 1. Internally, those derivative functions use the same acados integrator object. This implementation is a major contribution of this paper and available in the open-source software package casados-integrators with the permissive 2-clause BSD license [11].

When the CasadosIntegrator or one of its derivatives is called, all the options of the acados integrator are set, such that the requested sensitivities, respectively the function values, are computed with minimal overhead. When creating the acados integrator, all possible sensitivity options (forward, adjoint and Hessian) are activated, such that enough memory is allocated to execute all of them. However, some of those options are temporarily disabled in the wrapper when they are not needed.

E. *Limitations of the casados integrators*

When solving a multiple shooting OCP in CasADi, the integrator is called in the following order:

1) nominal call for each shooting gap

2) forward sensitivity call for each shooting gap
3) adjoint call + Hessian call for each shooting gap

This is suboptimal as the `acados` integrators offer to evaluate the nominal result and propagate sensitivity information within the same call, which allows to save some computations, such as the Newton iterations in case of an IRK. Moreover, this separate evaluation contradicts the concept of freezing all adaptive components within the integrator. Note that we observed no issues in this sense, in the numerical experiments, compared to other integrators, which have more adaptive components, such as `IDAS` and `CVODES`.

Additionally, the adjoint call before the Hessian call is unnecessary in case of the `acados`-based integrators. This call stems from the fact that some functions need the non-differentiated output for the derivative evaluation[1].

Possible extensions of this work include a similar Callback interface for MATLAB, an implementation using Python memoryviews to reduce the overhead of the proposed implementation and the possibility to use multiple integrators in parallel for the linearization of a multiple shooting based NLP. A number of breaking changes have been introduced in `CasADi` version 3.6.0 briefly before the final submission of this paper, which include major changes to its interface. The presented software is not yet adapted to this version.

IV. Numerical Experiments

The experiments presented here have been carried out using `acados` version v0.1.9 and `CasADi` 3.5.5 on a Lenovo T490s Laptop with Intel i5-8365U CPU, 16 GB RAM and Ubuntu 22.04. We compare the computation times of the `casados` integrator with the ones provided by `CasADi` by solving an NLP with a nonlinear hanging chain model. Second, we show how they can be used to solve a highly nonlinear NLP for an optimal orbit of an airborne wind energy system. All timings are recorded using the `CasADi` internal timer, for the NLP solution in Section IV-A and the function evaluations in IV-B respectively. The benchmark code, including all models, is publicly available at [11].

A. Nonlinear chain of masses

We regard the problem of controlling a nonlinear hanging chain of masses used in [16] and originally proposed in [17] in order to see how the CPU time of an NLP solution scales with the state dimension for different integrators. We can vary the number of masses in the chain and get a model with $n_\mathrm{x} = 6(n_\mathrm{mass} - 2) + 3$. We fix the horizon length to $N = 40$. The problem is solved using `IPOPT` and different integrators, namely `cacados` IRK and ERK with different wrappers and all available `CasADi` integrators, namely collocation (IRK), RK4 and CVODES, for the multiple shooting discretization. Additionally, the same problem is solved with a direct collocation discretization and `IPOPT` and `acados` SQP with a Gauß-Newton Hessian approximation. The settings of the different IRK version are equivalent and use the Gauß-Legendre methods. We note that all integrator settings

[1]`https://github.com/casadi/casadi/issues/2019`

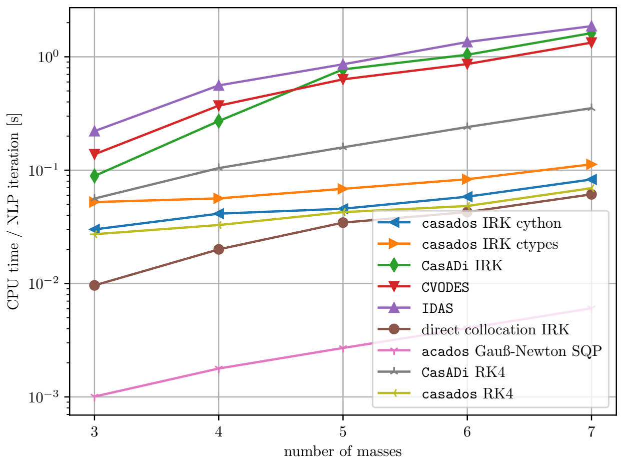

Fig. 2. CPU times with different integrators, IRK with $s = 2, n_\mathrm{steps} = 1$.

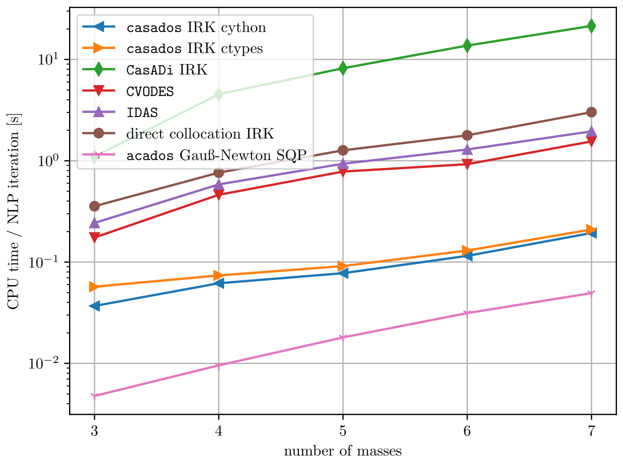

Fig. 3. CPU times with different integrators, IRK with $s = 4, n_\mathrm{steps} = 4$.

yield the same solution and the same number of NLP solver iterations. The computation times are shown in Figure 2 and 3, for IRK settings $s = 2, n_\mathrm{steps} = 1$ and $s = 4, n_\mathrm{steps} = 4$ respectively. Although it is understood that the accuracy of RK4 is sufficient for the present example, we use the computationally expensive setting $s = 4, n_\mathrm{steps} = 4$ to see how the computation times would evolve. The timings are compared in terms of CPU time per NLP iterations in seconds, while the number of NLP solver iterations was between 11 and 15 for the `IPOPT` versions and between 5 and 6 for the `acados` SQP solution for varying n_mass.

Figure 2 gives a fair comparison of RK4 and the different IRK versions, since they are all of order 4 and perform a single step. We note, that even for small state dimensions, the corresponding `casados` integrator outperforms the native RK4 method and that the speedup grows significantly with the state dimension. However, implicit methods still have better stability properties and are often preferred over explicit methods. Note that in the `acados` implementation the gap between the ERK and IRK method is much smaller compared to the one between the corresponding `CasADi` integrators. This can be attributed to the efficient `BLASFEO` linear algebra methods being used, as well as the tailored sensitivity

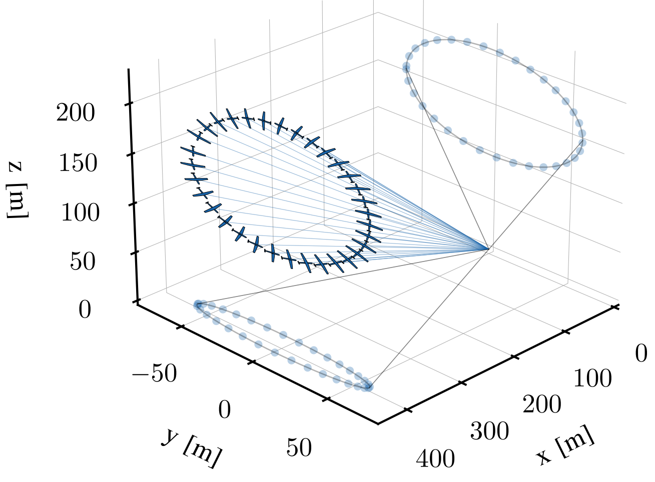

Fig. 4. Optimal position and orientation trajectory of the drag-mode AWE system. The wind vector is pointing along the positive x-axis. To improve visibility, the aircraft size is enhanced with a factor 3 in the plot.

propagation.

Comparing the differences between the two `casados` IRK integrators in Figure 2 and 3, we see that for a computationally intensive integration scheme, the difference of the two versions is insignificant. On the other hand, the speedup of the Cython wrapper is in the double digit percent range for simpler integration schemes, as in in Figure 2. Regarding Figure 3, we observe that the overall CPU time for the NLP solution with the equivalent IRK scheme is ≈ 100 times faster using the proposed integrator compared to the native `CasADi` IRK method.

Moreover, looking at the solution times of the equivalent direct collocation discretization, we observe that it is the fastest `IPOPT` variant overall for the simpler integration scheme in Figure 2, while being outperformed by a factor of 10 by the multiple shooting discretization using `casados` IRK in Figure 3.

The creation time, for a `casados` integrator with $n_{\text{mass}} = 7$ is 3.1s using `cython` and 0.8s using `ctypes`, showing that its use can pay of after a single NLP iteration even for $s = 2, n_{\text{steps}} = 1$.

The `acados` Gauß-Newton Hessian SQP algorithm consistently has the fastest iterations, due to multiple reasons. The OCP-NLP structure is fully exploited, no second-order sensitivities are evaluated (Gauß-Newton), no Python overhead of the integrators, only a single call to the integrator for the nominal simulation and sensitivity propagation per shooting node gap and SQP iteration (compare to Sec. III-E), fast QP solutions with `HPIPM`, `BLASFEO` and partial condensing [18] with a horizon of 10. While this is a lot faster than the `IPOPT casados` algorithm, the latter is numerically more robust and can find solutions for more challenging problems as shown in the next example.

B. Airborne Wind Energy (AWE) system

In the following, we consider the problem of finding a power-optimal periodic orbit for an airborne wind energy system. This type of system consists of a tethered aircraft that flies fast crosswind loops in order to extract energy from the wind. In this particular case, we consider a "drag-mode" system [19]. Here, electricity is generated on-board of the aircraft by means of small turbines that exert a braking force on the system, which is driven by the wind. The electricity is transported to a ground station through the tether.

	multiple shooting (`casados` IRK)	direct collocation
total	4.12	2.02
NLP function eval.	3.94	0.537
step computation	0.174	1.485
NLP solver iterations	20	10

TABLE I
AWE SYSTEM EXAMPLE: NLP SOLUTION TIMINGS IN S.

For this study, we choose the reference model presented in [20]. For the on-board turbine drag force and power, we use the expressions given in [21]. The model describes the full six-degree-of-freedom dynamics of a small AWE system with 5.5 m wing span aircraft. The system state is modeled in non-minimal coordinates so as to tailor the dynamic equations for use in Newton-type optimization algorithms. Hence, the resulting DAE dynamics contain six invariants, which are stabilized using a Baumgarte scheme [22]. The system dimensions are $n_{\text{x}} = 23, n_{\text{u}} = 4, n_{\text{z}} = 1$.

The dynamics are discretized using an IRK method with Gauß-Radau IIA Butcher tableau and $s = 4$, with sampling time $T_{\text{sim}} = 0.3364$ s. These discrete dynamics are used to formulate the periodic discrete-time optimal control problem as stated in [23, Eq. 3], with $N = 40$ intervals. We impose the realistic flight envelope constraints given in [24], resulting in 21 linear and 9 nonlinear inequality constraints per interval. The discrete stage cost is defined as the negative average on-board energy generated in one interval, combined with the regularization terms given in [24].

The resulting NLP is highly non-convex and requires a good initial guess to converge. Therefore we solve the problem based on an initial guess provided by the open-source AWE optimization toolbox `awebox` [25]. The power-optimal flight trajectory is visualized in Figure 4.

a) NLP solution timing comparison: Table I shows the NLP solution times for direct collocation and multiple shooting with `casados` IRK respectively, computed using `IPOPT` and `MA57`. Both methods result in timings in the same order of magnitude, with direct collocation outperforming multiple shooting by a factor of 2. While the multiple shooting timings are dominated by the NLP function evaluations, the step computation timings are reduced by one order of magnitude compared to direct collocation. This implies that the potential speedup of a parallelized linearization is significantly higher for the multiple shooting discretization.

b) Integrator Timing Comparison: Since we did not manage to solve the NLP described in the previous paragraph with the other integrators mentioned here despite trying various options, we limit the comparison of computation times to forward simulation and sensitivity propagation in this paragraph. We attribute the failure of the other integrators

	casados	CasADi IRK	speedup factor
median	0.2554	7.0845	27.74
max	0.4911	8.4917	17.29
min	0.2283	6.4680	28.33

TABLE II

TIMINGS IN MS, FORWARD SIMULATION OF GIVEN INITIAL STATE AND CONTROL TRAJECTORY.

	casados	CasADi IRK	speedup factor
median	0.6839	789.6	1154.6
max	0.9917	884.4	891.8
min	0.6528	744.4	1140.4

TABLE III

TIMING FOR JACOBIAN OF SIMULATION RESULT W.R.T. INITIAL STATE AND CONTROL INPUT FOR DIFFERENT VALUES.

to their adaptivity and inconsistency of sensitivities w.r.t. the nominal result in the solver, see Section II-C.In Table II, we compare the computation times of forward simulating the states and controls from the optimal trajectory in Figure 4. We observe that a speedup factor larger than 30 is achieved consistently.

In Table III, we compare the computation times of computing the Jacobian of the solution w.r.t. initial state and control input along the same trajectory. A comparison of those Jacobians shows that the maximum difference between any corresponding entries is below 10^{-12}, thereby confirming that the methods are mathematically equivalent. We note that for this kind of system the proposed integrator gives a speedup of roughly a factor 1000.

V. CONCLUSIONS & OUTLOOK

In this paper, we gave a detailed overview on the acados integrators and their interfaces to Python. We introduced a novel interface that makes those integrators available in CasADi Python via the open-source casados integrator software package. We show speedups of a factor of up to 1000 with respect to the integrators distributed by CasADi, when using them in an NLP solver. This enables the solution of problems that cannot be solved with the standard integrators in reasonable computation times. The most promising and cleanest direction for improving on this work, is a dedicated integrator implementation within CasADi, which could overcome the limitations mentioned.

ACKNOWLEDGMENTS

This research was supported by DFG via Research Unit FOR 2401 and project 424107692 and by the EU via ELO-X 953348. We thank Katrin Baumgärtner for fruitful discussions, Mario Zanon for asking for such a feature in the acados forum and Joris Gillis for advice regarding the Callback implementation.

REFERENCES

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[2] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[3] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, pp. 147–183, Oct 2021.

[4] G. Frison, T. Sartor, A. Zanelli, and M. Diehl, "The BLAS API of BLASFEO: Optimizing performance for small matrices," *ACM Transactions on Mathematical Software (TOMS)*, vol. 46, no. 2, pp. 15:1–15:36, 2020.

[5] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," in *Proceedings of the IFAC World Congress*, (Berlin, Germany), July 2020.

[6] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill, 2nd ed., 2017.

[7] R. Quirynen, B. Houska, and M. Diehl, "Symmetric hessian propagation for lifted collocation integrators in direct optimal control.," in *Proceedings of the American Control Conference (ACC)*, 2016.

[8] R. Quirynen, *Numerical Simulation Methods for Embedded Optimization*. PhD thesis, KU Leuven and University of Freiburg, 2017.

[9] J. Frey, R. Quirynen, D. Kouzoupis, G. Frison, J. Geisler, A. Schild, and M. Diehl, "Detecting and exploiting Generalized Nonlinear Static Feedback structures in DAE systems for MPC," in *Proceedings of the European Control Conference (ECC)*, 2019.

[10] R. Quirynen, S. Gros, B. Houska, and M. Diehl, "Lifted collocation integrators for direct optimal control in ACADO toolkit," *Mathematical Programming Computation*, vol. 9, no. 4, pp. 527–571, 2017.

[11] "casados integrators." https://github.com/FreyJo/casados-integrators, 2022.

[12] R. Serban and A. Hindmarsh, "CVODES: the sensitivity-enabled ODE solver in SUNDIALS," in *Proceedings of IDETC/CIE 2005*, 2005.

[13] A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker, and C. Woodward, "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 363–396, 2005.

[14] R. Quirynen, S. Gros, and M. Diehl, "Efficient NMPC for nonlinear models with linear subsystems," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 5101–5106, 2013.

[15] "openpilot." https://github.com/commaai/openpilot/.

[16] A. Zanelli, J. Frey, F. Messerer, and M. Diehl, "Zero-order robust nonlinear model predictive control with ellipsoidal uncertainty sets," *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*, 2021.

[17] L. Wirsching, H. G. Bock, and M. Diehl, "Fast NMPC of a chain of masses connected by springs," in *Proceedings of the IEEE International Conference on Control Applications, Munich*, 2006.

[18] G. Frison, D. Kouzoupis, J. B. Jørgensen, and M. Diehl, "An efficient implementation of partial condensing for nonlinear model predictive control," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 4457–4462, 2016.

[19] C. Vermillion, M. Cobb, L. Fagiano, R. Leuthold, M. Diehl, R. S. Smith, T. A. Wood, S. Rapp, R. Schmehl, D. Olinger, and M. Demetriou, "Electricity in the air: Insights from two decades of advanced control research and experimental flight testing of airborne wind energy systems," *Annual Reviews in Control*, vol. 52, 2021.

[20] E. C. Malz, J. Koenemann, S. Sieberling, and S. Gros, "A reference model for airborne wind energy systems for optimization and control," *Renewable Energy*, vol. 140, pp. 1004–1011, 2019.

[21] M. Zanon, S. Gros, J. Andersson, and M. Diehl, "Airborne wind energy based on dual airfoils," *IEEE Transactions on Control Systems Technology*, vol. 21, pp. 1215–1222, July 2013.

[22] S. Gros and M. Zanon, "Numerical optimal control with periodicity constraints in the presence of invariants," *IEEE Transactions on Automatic Control*, vol. 63, no. 9, pp. 2818–2832, 2018.

[23] J. De Schutter, M. Zanon, and M. Diehl, "TuneMPC, a tool for economic tuning of tracking (n)mpc problems," *IEEE Control Systems Letters*, vol. 4, no. 4, pp. 910–915, 2020.

[24] G. Licitra, J. Koenemann, A. Bürger, P. Williams, R. Ruiterkamp, and M. Diehl, "Performance assessment of a rigid wing airborne wind energy pumping system," *Energy*, vol. 173, pp. 569–585, 2019.

[25] "awebox: Modelling and optimal control of single- and multiple-kite systems for airborne wind energy." https://github.com/awebox, 2020.