Detecting and Exploiting Generalized Nonlinear Static Feedback Structures in DAE Systems for MPC

Jonathan Frey[1,5], Rien Quirynen[2], Dimitris Kouzoupis[1], Gianluca Frison[1], Jens Geisler[3], Axel Schild[4] and Moritz Diehl[1,5]

***Abstract*— Nonlinear Model Predictive Control (NMPC) and Moving Horizon Estimation (MHE) have become popular techniques for real-time control of various physical systems. However, in the context of embedded optimization, systems with fast dynamics still form a computational challenge. The simulation of the continuous time model and its sensitivity propagation are key tasks within any standard NMPC algorithm and have to be carried out frequently. We propose a general dynamic system structure that can handle all index-1 DAEs and enables us to exploit linear dependencies within the model. Additionally, we derive a tailored implicit Runge-Kutta (IRK) scheme that exploits this specific structure using a lifting-condensing approach and carrying out some computations offline. Moreover, we develop an algorithm to automatically transcribe continuous time models into this specific structure. The proposed method is applied to a wind turbine model, showing that the CPU time of the simulation and sensitivity propagation can be reduced by a factor greater than two.**

I. Introduction

Optimization based control techniques like NMPC [1] and MHE [2] enjoy a growing popularity. In the NMPC framework, numerical optimization methods, such as Interior Point (IP) or Sequential Quadratic Programming (SQP) methods, are applied to a Nonlinear Programming (NLP) formulation which arises from an Optimal Control Problem (OCP) via discretization. The simulation of a dynamic model and sensitivity propagation, (i.e. computing the derivative of the simulation result with respect to the initial state and control) are key tasks that have to be carried out often within high level optimal control algorithms. Thus, it is critical to have efficient integration schemes with sensitivity propagation available that can be called by the high level algorithms and which we refer to as *integrators*.

For continuous time dynamic models that are formulated implicitly or in which stiff modes are present, it is recommended to use implicit integration schemes as they have better stability properties and can typically provide similar results at lower computational cost for these types of models. Implicit Runge-Kutta (IRK) methods are one of the most famous and widely used class of implicit integration schemes, [3], and form the main focus of this paper. Let us introduce the following differential algebraic equation (DAE) model

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

where $x(\cdot): \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$ are the differential states, $\dot{x}(\cdot): \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$ their time-derivatives, $u(\cdot): \mathbb{R} \to \mathbb{R}^{n_\mathrm{u}}$ the control inputs and $z(\cdot): \mathbb{R} \to \mathbb{R}^{n_\mathrm{z}}$ the algebraic states. The DAE in (1) is called index-1 if $\frac{\partial f^{\mathrm{impl}}}{\partial \dot{x}, z}(\cdot) \in \mathbb{R}^{(n_\mathrm{x}+n_\mathrm{z})\times(n_\mathrm{x}+n_\mathrm{z})}$ is a regular matrix. Given the initial state value x_n, the applied control input u_n and the integration step size T_{int}, one step of an IRK method applied to this model can be written as

$$0 = f^{\mathrm{impl}}\left(k_i, x_n + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j, z_i, u_n\right), \forall i = 1, \dots, s \tag{2a}$$

$$x_{n+1} = x_n + T_{\mathrm{int}} \sum_{j=1}^{s} b_j k_j. \tag{2b}$$

Hereby, the integration variables k_i, z_i denote the stage values for the $\dot{x}(t)$, $z(t)$ trajectory at the times $t_{n,i} = t_n + c_i T_{\mathrm{int}}$ and the coefficients a_{ij}, b_j, c_j, for $i, j = 1, \dots, s$ define the Runge-Kutta method. The triple $(A_{\mathrm{but}}, b_{\mathrm{but}}, c_{\mathrm{but}})$ is generally referred to as the *Butcher tableau* and s is the *number of stages*. IRK schemes require solving the system of nonlinear equations in Eq. (2a), which is usually done by performing a certain number of Newton-type iterations, [4].

Typically, the main computational cost in a classical IRK scheme is the factorization of the Newton matrix, which is of dimension $s(n_\mathrm{x} + n_\mathrm{z})$. When using dense linear algebra routines, this matrix factorization results in a computational cost of order $s^3(n_\mathrm{x} + n_\mathrm{z})^3$.

One classic approach to reduce the computational cost is to use a structured Jacobian approximation within an inexact Newton scheme, in which the derivative of f^{impl} is evaluated only once in combination with a tailored transformation of the linear system, as proposed in [5]. A more recent approach is to exploit structures commonly present within dynamic system models [6], [7]. The latter approach allows the use of exact Jacobian evaluations and it can be used even for relatively low order IRK formulas.

In this paper, we present a structured dynamic system and an IRK scheme tailored for this kind of system, that are further generalizations of the previously presented structured dynamic system and associated IRK schemes in [6], [7]. Furthermore, we present an algorithm to automatically transcribe continuous time models into the presented structured dynamic system.

*This research was supported by the Federal Ministry for Economic Affairs and Energy (BMWi) via eco4wind (0324125B) and DyConPV (0324166B), and by DFG via Research Unit FOR 2401.

[1]Department IMTEK, University of Freiburg, Georges-Koehler-Allee 102, 79110 Freiburg, Germany.

[2]Mitsubishi Electric Research Laboratories, Cambridge, MA, 02139, USA.

[3]Senvion GmbH, Überseering 10, 22297 Hamburg, Germany.

[4]IAV GmbH, Rockwellstr. 16, 38518 Gifhorn, Germany

[5]Department of Mathematics, University of Freiburg, Germany

Both the presented integration scheme and the automatic transcription method have been implemented within the `acados` framework and are part of the open-source Github repository [8]. `acados` is a recently developed modular open-source software package for embedded optimization [9]. It gathers a variety of numerical algorithms for NMPC that are implemented in `C` using the efficient basic linear algebra library BLASFEO [10], [11] to perform linear algebra operations efficiently on various CPU architectures.

The paper is organized as follows: Section II presents and motivates the considered structured dynamic system. In Section III, the structure exploiting IRK scheme is presented. Section IV briefly summarizes the model transcription algorithm. Section V presents some numerical results applying the integration scheme to a complex wind turbine model. Section VI concludes the paper.

II. General Structured Dynamic System

The structured dynamic system model that will be considered in the remainder of this paper reads as follows:

$$E\begin{bmatrix}\dot{x}^{[1]}\\ z^{[1]}\end{bmatrix} = Ax^{[1]} + Bu + C\phi(\underbrace{L_{\dot{x}}\dot{x}^{[1]} + L_{x}x^{[1]} + L_{z}z^{[1]}}_{=:y}, \underbrace{L_{u}u}_{=:\hat{u}}) + c \tag{3a}$$

$$E^{\mathrm{LO}}\begin{bmatrix}\dot{x}^{[2]}\\ z^{[2]}\end{bmatrix} = A^{\mathrm{LO}}x^{[2]} + f_{\mathrm{LO}}(\dot{x}^{[1]}, x^{[1]}, z^{[1]}, u). \tag{3b}$$

We further refer to (3) as the *Generalized Nonlinear Static Feedback structure* (GNSF) as it couples a nonlinear static feedback (NSF) type system in (3a) [7] with a linear output system (LOS) in (3b), [6]. For definitions, we denote the differential states as $x = \left[x^{[1]\top}, x^{[2]\top}\right]^{\top} \in \mathbb{R}^{n_{\mathrm{x}}}$, $x^{[1]} \in \mathbb{R}^{n_{\mathrm{x}_1}}$, $x^{[2]} \in \mathbb{R}^{n_{\mathrm{x}_2}}$, the algebraic variables $z = \left[z^{[1]\top}, z^{[2]\top}\right]^{\top} \in \mathbb{R}^{n_{\mathrm{z}}}$, $z^{[1]} \in \mathbb{R}^{n_{\mathrm{z}_1}}$, $z^{[2]} \in \mathbb{R}^{n_{\mathrm{z}_2}}$, the controls $u \in \mathbb{R}^{n_{\mathrm{u}}}$ and the (nonlinear) functions $\phi : \mathbb{R}^{n_{\mathrm{y}}+n_{\hat{\mathrm{u}}}} \to \mathbb{R}^{n_{\mathrm{out}}}$, $f_{\mathrm{LO}} : \mathbb{R}^{2n_{\mathrm{x}_1}+n_{\mathrm{z}_1}+n_{\mathrm{u}}} \to \mathbb{R}^{n_{\mathrm{x}_2}}$, which will be referred to as the *nonlinearity* and *linear output* functions respectively. Furthermore, we use the model matrices $L_{\dot{\mathrm{x}}}, L_{\mathrm{x}} \in \mathbb{R}^{n_{\mathrm{y}} \times n_{\mathrm{x}_1}}$, $L_{\mathrm{z}} \in \mathbb{R}^{n_{\mathrm{y}} \times n_{\mathrm{z}_1}}$, $L_{\mathrm{u}} \in \mathbb{R}^{n_{\hat{\mathrm{u}}} \times n_{\mathrm{u}}}$, which we refer to as *linear input* matrices, and the matrices $A \in \mathbb{R}^{(n_{\mathrm{x}_1}+n_{\mathrm{z}_1}) \times n_{\mathrm{x}_1}}$, $B \in \mathbb{R}^{(n_{\mathrm{x}_1}+n_{\mathrm{z}_1}) \times n_{\mathrm{u}}}$, $C \in \mathbb{R}^{(n_{\mathrm{x}_1}+n_{\mathrm{z}_1}) \times n_{\mathrm{out}}}$, $E \in \mathbb{R}^{(n_{\mathrm{x}_1}+n_{\mathrm{z}_1}) \times (n_{\mathrm{x}_1}+n_{\mathrm{z}_1})}$ and the vector $c \in \mathbb{R}^{n_{\mathrm{x}_1}+n_{\mathrm{z}_1}}$, which are constant and can be split naturally as

$$E = \begin{bmatrix}E_{11} & E_{12}\\ E_{21} & E_{22}\end{bmatrix}\ A = \begin{bmatrix}A_1\\ A_2\end{bmatrix}\ B = \begin{bmatrix}B_1\\ B_2\end{bmatrix}\ C = \begin{bmatrix}C_1\\ C_2\end{bmatrix}\ c = \begin{bmatrix}c_1\\ c_2\end{bmatrix}$$

to denote the terms corresponding to the differential and algebraic equations in (3a). Finally, the matrices $A^{\mathrm{LO}} \in \mathbb{R}^{(n_{\mathrm{x}_2}+n_{\mathrm{z}_2}) \times n_{\mathrm{x}_2}}$ and $E^{\mathrm{LO}} \in \mathbb{R}^{(n_{\mathrm{x}_2}+n_{\mathrm{z}_2}) \times (n_{\mathrm{x}_2}+n_{\mathrm{z}_2})}$ are used for the LOS in (3b).

In the implementation of a standard IRK scheme on (3), a square matrix of size $s(n_{\mathrm{x}} + n_{\mathrm{z}})$ needs to be factorized for each Newton-type iteration on the nonlinear system in (2a). In the next section of this paper, an equivalent structure exploiting IRK scheme is presented in which the matrix to be factorized is of dimension sn_{out} instead, where always $n_{\mathrm{out}} \le (n_{\mathrm{x}} + n_{\mathrm{z}})$ but often even $n_{\mathrm{out}} \ll (n_{\mathrm{x}} + n_{\mathrm{z}})$ holds.

A. Assumptions on GNSF Dynamic Model

We assume that $E - C\frac{\partial\phi}{\partial y}[L_{\dot{\mathrm{x}}}, L_{\mathrm{z}}]$ and E^{LO} are invertible, such that $\dot{x}, z$ have well-defined trajectories, [3]. In order to apply the IRK scheme derived in the next section, we have to additionally assume that E is invertible.

B. Motivation for GNSF Dynamic Model Formulation

The main goal of this work was to further generalize structured linear dependencies in nonlinear dynamic models such that the corresponding IRK scheme can exploit both of the previously proposed structures in [6], [7] as well as the schemes that are tailored to them. More specifically, we aim to reduce the dimensions of the matrix in the Newton-type iterations even further by exploiting multiple of these model structures combined. We refer to [12] for an extensive discussion on why this is indeed the case for our proposed formulation.

Furthermore, we have extended the previous structures to support also index-1 DAEs, as suggested in [7]. GNSF maximizes the linear dependencies in both subsystems, preserving the property that the NSF system in (3a) can be simulated independently of the LOS (3b), which is essential for the proposed algorithm in Section III.
Additionally, we chose to split the input of ϕ into y and $\hat{u}$ to evaluate the derivatives of ϕ only with respect to the variables needed within the Newton iterations, namely $x, \dot{x}, z$.

III. GNSF-Structure Exploiting IRK Scheme

In this Section, we derive the GNSF structure exploiting IRK scheme (GNSF-IRK) in which the computational complexity of a Newton step is of order $s^3 n_{\mathrm{out}}^3$ instead of the standard computational cost of order $s^3(n_{\mathrm{x}} + n_{\mathrm{z}})^3$.
For notational convenience, we denote the integration variables for each of the two subsystems as

$$K^{[1]} = \left(k_1^{[1]\top}, \ldots, k_s^{[1]\top}\right)^{\top}, \quad Z^{[1]} = \left(z_1^{[1]\top}, \ldots, z_s^{[1]\top}\right)^{\top},$$
$$K^{[2]} = \left(k_1^{[2]\top}, z_1^{[2]\top}, \ldots, k_s^{[2]\top}, z_s^{[2]\top}\right)^{\top},$$

where $k_i^{[j]}, z_i^{[j]}$ discretize the $\dot{x}^{[j]}, z^{[j]}$ trajectory for $j = 1, 2$.
As mentioned before, the NSF type system (3a) can be simulated independently from the LOS in (3b).

A. Simulation of the NSF subsystem

Let us now apply the IRK scheme (2) to the NSF subsystem in (3a). Using the above abbreviations, we are able to write the integration equation (2a) as

$$0 = \overbrace{\begin{bmatrix}\mathbf{E}_1 & -\mathbf{D}_1\\ -\mathbf{D}_2 & \mathbf{E}_2\end{bmatrix}}^{\mathbf{E}}\begin{bmatrix}K^{[1]}\\ Z^{[1]}\end{bmatrix} - \overbrace{\begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix}}^{\mathbf{A}}x_n^{[1]} - \overbrace{\begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix}}^{\mathbf{B}}u_n - \overbrace{\begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix}}^{\mathbf{c}}$$
$$- \underbrace{\begin{bmatrix}\mathbf{C}_1\\ \mathbf{C}_2\end{bmatrix}}_{\mathbf{C}}\Phi(\mathbf{L}_{\mathrm{K}}K^{[1]} + \mathbf{L}_{\mathrm{x}}x_n^{[1]} + \mathbf{L}_{\mathrm{Z}}Z^{[1]}, L_{\mathrm{u}}u_n) \tag{4}$$

We introduce the matrices $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{D}_1$, $\mathbf{D}_2, \mathbf{E}_1, \mathbf{E}_2$, using $\otimes$ to denote the Kronecker product:

$$\begin{aligned}
&\mathbf{A}_1 = \begin{bmatrix} A_1 \\ \dots \\ A_1 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} A_2 \\ \dots \\ A_2 \end{bmatrix}, \mathbf{B}_1 = \begin{bmatrix} B_1 \\ \dots \\ B_1 \end{bmatrix}, \mathbf{B}_2 = \begin{bmatrix} B_2 \\ \dots \\ B_2 \end{bmatrix}, \\
&\mathbf{C}_1 = \mathbb{1}_s \otimes C_1, \quad \mathbf{C}_2 = \mathbb{1}_s \otimes C_2, \quad \mathbf{D}_1 = -\mathbb{1}_s \otimes E_{12}, \\
&\mathbf{D}_2 = T_{\text{int}} A_{\text{but}} \otimes A_2 - \mathbb{1}_s \otimes E_{21}, \\
&\mathbf{E}_1 = \mathbb{1}_s \otimes E_{11} - T_{\text{int}} A_{\text{but}} \otimes A_1, \quad \mathbf{E}_2 = \mathbb{1}_s \otimes E_{22}.
\end{aligned} \tag{5}$$

The matrices $\mathbf{L}_\text{Z}, \mathbf{L}_\text{K}, \mathbf{L}_\text{x}$ and the vectors $\mathbf{c}_1, \mathbf{c}_2$ read as

$$\begin{aligned}
&\mathbf{L}_\text{Z} = \mathbb{1}_s \otimes L_\text{z}, \quad \mathbf{L}_\text{K} = T_{\text{int}} A_{\text{but}} \otimes L_\text{x} + \mathbb{1}_s \otimes L_{\dot{\text{x}}}, \\
&\mathbf{L}_\text{x} = \begin{bmatrix} L_\text{x} \\ \dots \\ L_\text{x} \end{bmatrix}, \quad \mathbf{c}_1 = \begin{bmatrix} c_1 \\ \dots \\ c_1 \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} c_2 \\ \dots \\ c_2 \end{bmatrix}.
\end{aligned} \tag{6}$$

In addition, we define the function $\Phi : \mathbb{R}^{sn_\text{in}} \to \mathbb{R}^{sn_\text{out}}$ as

$$\Phi(\mathbf{y}, u) = \begin{bmatrix} \phi(y_1, L_\text{u} u) \\ \dots \\ \phi(y_s, L_\text{u} u) \end{bmatrix}, \text{ for } \mathbf{y} = \begin{bmatrix} y_1 \\ \dots \\ y_s \end{bmatrix} \in \mathbb{R}^{sn_\text{in}}. \tag{7}$$

Now, we apply a lifting-condensing technique similar to the one suggested in [7], rewriting Eq. (4) as

$$\mathbf{E} \begin{bmatrix} K^{[1]} \\ Z^{[1]} \end{bmatrix} - \mathbf{A} x_n^{[1]} - \mathbf{B} u_n - \mathbf{c} - \mathbf{C} \mathbf{v} = 0 \tag{8a}$$

$$\mathbf{v} - \Phi(\mathbf{L}_\text{x} x_n^{[1]} + \mathbf{L}_\text{K} K^{[1]} + \mathbf{L}_\text{Z} Z^{[1]}, L_\text{u} u_n) = 0. \tag{8b}$$

We want to argue that the matrix $\mathbf{E}$ is invertible for sufficiently small step sizes T_int. For $T_\text{int} = 0$, we note that $\mathbf{E}$ can be obtained from $\mathbb{1}_s \otimes E$ by permuting its rows and columns, thus they are equivalent. Since the matrix E is invertible, $\mathbb{1}_s \otimes E$ and $\mathbf{E}$ are invertible.
This enables us to rearrange (8a) as

$$\begin{bmatrix} K^{[1]} \\ Z^{[1]} \end{bmatrix} = \mathbf{E}^{-1} (\mathbf{A} x_n^{[1]} + \mathbf{B}_1 u_n + \mathbf{C} \mathbf{v} + \mathbf{c}) \tag{9}$$

Within the GNSF-IRK scheme, we want to eliminate the variables $K^{[1]}, Z^{[1]}$ and only keep $\mathbf{v}, x_n^{[1]}, u_n$. We can obtain the IRK integration variables $W^\top = \begin{bmatrix} K^{[1]^\top}, Z^{[1]^\top} \end{bmatrix}$ as linear transformations of $\mathbf{v}, x_n^{[1]}, u_n$.

$$W = \mathbf{W}_\mathbf{v} \mathbf{v} + \mathbf{W}_\text{u} u_n + \mathbf{W}_\text{x} x_n^{[1]} + \mathbf{W}_0, \tag{10}$$

where we introduced the matrices $\mathbf{W}_\mathbf{v}$, $\mathbf{W}_\text{u}$, $\mathbf{W}_\text{x}$ and $\mathbf{W}_0$ that are computed as

$$\begin{aligned}
\mathbf{W}_\mathbf{v} &= \mathbf{E}^{-1} \mathbf{C}, \quad \mathbf{W}_\text{u} = \mathbf{E}^{-1} \mathbf{B} \\
\mathbf{W}_\text{x} &= \mathbf{E}^{-1} \mathbf{A}, \quad \mathbf{W}_0 = \mathbf{E}^{-1} \mathbf{c}
\end{aligned} \tag{11}$$

We split the $\mathbf{W}_*$ matrices in the first sn_{x_1} and the remaining sn_{z_1} rows

$$\mathbf{W}_\mathbf{v} = \begin{bmatrix} \mathbf{K}_\mathbf{v} \\ \mathbf{Z}_\mathbf{v} \end{bmatrix}, \mathbf{W}_\text{u} = \begin{bmatrix} \mathbf{K}_\text{u} \\ \mathbf{Z}_\text{u} \end{bmatrix}, \mathbf{W}_\text{x} = \begin{bmatrix} \mathbf{K}_\text{x} \\ \mathbf{Z}_\text{x} \end{bmatrix}, \mathbf{W}_0 = \begin{bmatrix} \mathbf{K}_0 \\ \mathbf{Z}_0 \end{bmatrix}$$

such that we can obtain $K^{[1]}$ and $Z^{[1]}$ separately as follows

$$K^{[1]} = \mathbf{K}_\mathbf{v} \mathbf{v} + \mathbf{K}_\text{u} u_n + \mathbf{K}_\text{x} x_n^{[1]} + \mathbf{K}_0, \tag{12a}$$

$$Z^{[1]} = \mathbf{Z}_\mathbf{v} \mathbf{v} + \mathbf{Z}_\text{u} u_n + \mathbf{Z}_\text{x} x_n^{[1]} + \mathbf{Z}_0, \tag{12b}$$

Finally, we derive an expression for the input argument of the gathered nonlinearity function Φ, which reads as

$$\begin{aligned}
\mathbf{y} &= \mathbf{L}_\text{x} x_n^{[1]} + \mathbf{L}_\text{K} K^{[1]} + \mathbf{L}_\text{Z} Z^{[1]} \\
&= \mathbf{Y}_\text{x} x_n^{[1]} + \mathbf{Y}_\text{u} u_n + \mathbf{Y}_\mathbf{v} \mathbf{v} + \mathbf{Y}_0,
\end{aligned} \tag{13}$$

where the matrices $\mathbf{Y}_\mathbf{v}, \mathbf{Y}_\text{u}, \mathbf{Y}_\text{x}, \mathbf{Y}_0$ are defined below

$$\begin{aligned}
\mathbf{Y}_\text{x} &= \mathbf{L}_\text{x} + \mathbf{L}_\text{K} \mathbf{K}_\text{x} + \mathbf{L}_\text{Z} \mathbf{Z}_\text{x} \\
\mathbf{Y}_\text{u} &= \mathbf{L}_\text{K} \mathbf{K}_\text{u} + \mathbf{L}_\text{Z} \mathbf{Z}_\text{u} \\
\mathbf{Y}_\mathbf{v} &= \mathbf{L}_\text{K} \mathbf{K}_\mathbf{v} + \mathbf{L}_\text{Z} \mathbf{Z}_\mathbf{v} \\
\mathbf{Y}_0 &= \mathbf{L}_\text{K} \mathbf{K}_0 + \mathbf{L}_\text{Z} \mathbf{Z}_0.
\end{aligned} \tag{14}$$

All these bold matrices can be precomputed offline, but only after the choice of the integrator options, i.e., the Butcher tableau and the integration step size T_int.
The nonlinear part of the integration equations can be gathered in the residual function $\mathbf{r}$, on which we will perform Newton-type iterations in the main part of the simulation:

$$\mathbf{r}(\mathbf{v}, x_n^{[1]}, u_n) = \mathbf{v} - \Phi(\mathbf{Y}_\text{x} x_n^{[1]} + \mathbf{Y}_\text{u} u_n + \mathbf{Y}_\mathbf{v} \mathbf{v} + \mathbf{Y}_0, L_\text{u} u_n) = 0. \tag{15}$$

A full-step Newton iteration then reads as:

$$\begin{aligned}
&\mathbf{v} \leftarrow \mathbf{v} + \Delta\mathbf{v}, \\
&\text{where} \quad \Delta\mathbf{v} = -\left(\frac{\partial \mathbf{r}}{\partial \mathbf{v}} (\mathbf{v}, x_n^{[1]}, u_n) \right)^{-1} \mathbf{r}(\mathbf{v}, x_n^{[1]}, u_n).
\end{aligned} \tag{16}$$

Therefore, we need the Jacobian $\frac{\partial \mathbf{r}}{\partial \mathbf{v}}$ and later on, for the purpose of sensitivity analysis, we additionally need the derivatives with respect to the control inputs and the initial state value, i.e., $\frac{\partial \mathbf{r}}{\partial (\mathbf{v}, x_n^{[1]}, u_n)}$. They can be computed as

$$\begin{aligned}
\frac{\partial \mathbf{r}}{\partial \mathbf{v}} (\cdot) &= \mathbb{1} - \frac{\partial \Phi}{\partial \mathbf{y}} (\cdot) \mathbf{Y}_\mathbf{v} \\
\frac{\partial \mathbf{r}}{\partial (x_n^{[1]}, u_n)} (\cdot) &= -\frac{\partial \Phi}{\partial \mathbf{y}} (\cdot) \begin{bmatrix} \mathbf{Y}_\text{x} & \mathbf{Y}_\text{u} \end{bmatrix} - \begin{bmatrix} \mathbb{0} & \frac{\partial \Phi}{\partial \hat{u}} (\cdot) L_\text{u} \end{bmatrix},
\end{aligned} \tag{17}$$

where

$$\frac{\partial \Phi}{\partial \mathbf{y}} (\mathbf{y}, u) = \operatorname{diag} \left(\frac{\partial \phi}{\partial y} (y_i, u) \right)_{i=1,\dots,s}. \tag{18}$$

Note, that in the `acados` implementation $\frac{\partial \phi}{\partial y}$ is evaluated by a `C` function. The model functions are typically code generated using `CasADi`, a tool that started as an algorithmic differentiation (AD) tool and grew into the direction of modeling and optimal control, [13].

B. Simulation of the Linear Output System

Now we can solve the IRK equations (2a) corresponding to the linear output system (3b). This can be written as

$$G_2(w_n, K^{[1]}, Z^{[1]}, K^{[2]}) = 0, \tag{19}$$

where the function G_2 is defined as

$$G_2(w_n, K^{[1]}, Z^{[1]}, K^{[2]}) := \left[E^{\mathrm{LO}} \begin{bmatrix} k_i^{[2]} \\ z_i^{[2]} \end{bmatrix} - A^{\mathrm{LO}}(x_n^{[2]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[2]}) - f_{\mathrm{LO}}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u_n) \right]_{i=1,\dots,s}. \tag{20}$$

Because Eq. (19) is linear in $K^{[2]}$, the matrix $M_2 = \frac{\partial G_2}{\partial K^{[2]}}$ is constant and the integration variables $K^{[2]}$ can be obtained exactly by solving one linear system. This equation reads as $K^{[2]} = -M_2^{-1} G_2(w_n, K^{[1]}, Z^{[1]}, 0)$. The Jacobian matrix can be computed and factorized offline as:

$$M_2 = \mathbb{1}_s \otimes E^{\mathrm{LO}} - T_{\mathrm{int}} A_{\mathrm{but}} \otimes \begin{bmatrix} A^{\mathrm{LO}} & \mathbb{0}_{(n_{\mathrm{x}_2}+n_{\mathrm{z}_2}) \times n_{\mathrm{z}_2}} \end{bmatrix}. \tag{21}$$

C. First Order Sensitivity Propagation

The sensitivities $\frac{\partial \mathbf{v}}{\partial x^{[1]}, u}$ can be obtained by applying the implicit function theorem on Eq. (15), yielding the following system of linear equations

$$\frac{\partial \mathbf{r}}{\partial \mathbf{v}} \cdot \frac{\partial \mathbf{v}}{\partial (x^{[1]}, u)} = - \left(\frac{\partial \mathbf{r}}{\partial (x^{[1]}, u)} \right). \tag{22}$$

Hereby, the Jacobian matrix to be factorized is the same as in the simulation part, (see Eqs. (16),(17)).
For the sensitivities of $K^{[1]}$ and $Z^{[1]}$, we use (12a) and (12b) respectively to directly obtain

$$\frac{\partial K^{[1]}}{\partial w_n} = \left[\mathbf{K}_{\mathrm{x}} + \mathbf{K}_{\mathrm{v}} \frac{\partial \mathbf{v}}{\partial x_n^{[1]}},\ \mathbb{0},\ \mathbf{K}_{\mathrm{u}} + \mathbf{K}_{\mathrm{v}} \frac{\partial \mathbf{v}}{\partial u} \right] \tag{23a}$$

$$\frac{\partial Z^{[1]}}{\partial w_n} = \left[\mathbf{Z}_{\mathrm{x}} + \mathbf{Z}_{\mathrm{v}} \frac{\partial \mathbf{v}}{\partial x_n^{[1]}},\ \mathbb{0},\ \mathbf{Z}_{\mathrm{u}} + \mathbf{Z}_{\mathrm{v}} \frac{\partial \mathbf{v}}{\partial u} \right], \tag{23b}$$

using the shorthand $w_n = \left[x_n^\top, u_n^\top \right]^\top$ to expand the sensitivities to the original space. Note that, in an efficient implementation, storing the zero matrices $\frac{\partial K^{[1]}}{\partial x_n^{[2]}}, \frac{\partial Z^{[1]}}{\partial x_n^{[2]}}$ is avoided.
Now, looking at the $K^{[2]}$ variables, we apply the implicit function theorem on Eq. (19) and obtain

$$\frac{\partial K^{[2]}}{\partial w_n} = -M_2^{-1} \left[\frac{\partial G_2}{\partial w_n} + \frac{\partial G_2}{\partial K^{[1]}} \frac{\partial K^{[1]}}{\partial w_n} + \frac{\partial G_2}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial w_n} \right]. \tag{24}$$

The implementation of the GNSF-IRK scheme in `acados` additionally comes with an efficient way to propagate the adjoint sensitivities and to evaluate sensitivities of the algebraic variables, (see [12]). Second order sensitivity propagation is a feature that is left for future development.

IV. Automatic GNSF Model Transcription

A. Principles for the Model Transcription

For an efficient deployment of GNSF-IRK, the given dynamic model has to be transcribed into the GNSF format (3). This transcription is neither trivial nor unique. When transcribing a dynamic system model into the GNSF structure, the following principles should be kept in mind to get a good speedup by the structure exploitation:

1) **Maximize components modeled in LOS** (3b)**:** For each component of the differential (respectively algebraic) state vector x (z), one has to decide if it is made part of the NSF type system (3a) or the LOS in (3b). Since the simulation of the LOS is computationally cheaper, one should make a component always part of the LOS if possible. If this is not possible, i.e., if the state depends nonlinearly on itself or any other state variable in the NSF depends on it, the component must be made part of the NSF type system.
2) **Minimize dimensions of Newton matrix:** The output dimension n_{out} of the nonlinearity function ϕ should be made as small as possible, as the computational complexity is cubic in n_{out}.
3) **Minimize input dimension of ϕ:** The input dimensions $n_{\mathrm{y}}, n_{\hat{\mathrm{u}}}$ of ϕ should be as small as possible to minimize the size of the linear input matrices $L_{\dot{\mathrm{x}}}, L_{\mathrm{x}}, L_{\mathrm{z}}, L_{\mathrm{u}}$ and the cost of multiplying with $\mathbf{Y}_{\mathrm{v}}, \mathbf{Y}_{\mathrm{u}}, \mathbf{Y}_{\mathrm{x}}$ (14) and L_{u}, which has to be done in each Newton iteration and within the sensitivity propagation.

B. An Automatic Transcription Method

In order to make a structure exploiting integrator conveniently usable, it is crucial to provide an algorithm which automatically transcribes a dynamic system into the specific structure used in the IRK implementation. As most continuous-time models can be brought into the form of an implicit index-1 DAE relatively easily, it is desirable to have an algorithm that transcribes this kind of model (1) into the GNSF structure (3) and follows the principles 1 to 3 from the previous subsection. One possible algorithm that achieves this is described briefly in the rest of this section. A detailed description can be found in [12].

First, the algorithm creates a trivial equivalent GNSF formulation, in which $A, B, E, c = \mathbb{0}$, $C = \mathbb{1}$ and $\phi = f^{\mathrm{impl}}$, such that equivalence of the models can be repeatedly checked. Subsequently, it successively detects all affine linear terms, subtracts them from the expression ϕ and adds corresponding entries to the model matrices in (3a). Afterwards, the algorithm checks if ϕ has constant zero entries. These entries will be removed and the matrix C is adapted accordingly. Then, the algorithm analyzes the equations and detects a maximal set of x, z-components (and equations) that can be modeled by the LOS (3b). These equations will be reformulated in the form (3b) and removed from the NSF part. Finally, it checks if the transcribed model is valid for GNSF-IRK (see paragraph II-A). If this is not the case, the algorithm proceeds by trying to fulfill the requirements adding ones on the diagonal E, E^{LO} and modify C, ϕ accordingly.

The transcription algorithm is implemented as a MATLAB function for `CasADi` models using the `SX` symbolic variables. It was made part of the `acados` Github repository [8].

V. Numerical Experiments

The application of NMPC on wind turbines has been investigated already from various points of view [14], [15]. In this section, we want to briefly present a medium size wind turbine model (Section V-A) and discuss some numerical properties of the presented integration scheme by applying it to this model (Section V-B).

A. Simplified Multibody Wind Turbine Model

The model is derived from a simplified flexible multibody formulation. The differential states of the model are denoted by $x = \left[q^\top, \dot{q}^\top, \vartheta\right]^\top \in \mathbb{R}^{13}$, whereby ϑ corresponds to the pitch angle and $q \in \mathbb{R}^6$ consists of the first tower eigenmodes in x and y direction, the rotation angle of the hub, the collective deformation of all three blades in their first mode in tangential direction and in axial direction, and the rotation angle of the generator. The control vector $u = \left(T_{\text{gen}}, \vartheta_{\text{ref}}\right)^\top \in \mathbb{R}^2$ consists of the generator torque and the pitch angle that is tracked by a lower level controller.
The first part of the model equations enforce $\frac{\partial q}{\partial t} = \dot{q}$, the second part of the implicit ODE is presented in Eq. (26) where we use the shorthands

$$\begin{aligned} v_{\text{rot\_rel}} &= \dot{q}_1 + p_{30}\,\dot{q}_4 \cos\vartheta, \\ v_{\text{wind\_eff}} &= v_{\text{wind}} - v_{\text{rot\_rel}}, \\ \lambda &= \dot{q}_3\, p_{33}/v_{\text{wind}}, \end{aligned} \tag{26}$$

to define

$$\begin{aligned} P_{\text{rot}} &= 0.5 p_{32}\; {v_{\text{wind\_eff}}}^3\, c_{\text{p}}(\lambda, \vartheta) p_{31}, \\ T_{\text{rot}} &= P_{\text{rot}} / \dot{q}_3, \\ F_{\text{thrust}} &= 0.5 p_{32}\; {v_{\text{wind\_eff}}}^2\, c_{\text{t}}(\lambda, \vartheta). \end{aligned} \tag{27}$$

The parameter values p_i for $i = 1, \ldots, 34$ need to be kept confidential for the moment. The dynamics of ϑ read as

$$\dot{\vartheta} = (\vartheta_{\text{ref}} - \vartheta)/p_{34}. \tag{28}$$

The aerodynamic coefficients $c_{\text{p}}, c_{\text{t}}$ are modeled as polynomials that are well defined only in a certain region, in which the simulations in V-B stay. Alternatively, it is common to use splines for the aerodynamic coefficients, [16]. However, the splines are not yet compatible with `CasADi SX` symbolic variables and the automatic transcription method.
The wind speed v_{wind} is considered as a parameter that is set to $12\frac{\text{m}}{\text{s}}$. We applied the automatic transcription method from Section IV to the model equations and obtained a GNSF-structured dynamic model with dimensions listed in Table I.

TABLE I
Model dimensions

Original model dimensions				GNSF dimensions				
n_{x}	n_{u}	n_{z}	n_{param}	n_{x_1}	n_{z_1}	n_{out}	n_{y}	$n_{\hat{\text{u}}}$
13	2	0	1	11	0	5	8	1

B. Numerical Simulation Results

We applied both a standard IRK method and the presented GNSF-IRK scheme with forward sensitivity generation to the dynamic model from V-A with a simulation horizon of 0.2s and a constant control input to obtain computational results that will be discussed in this section. Both schemes are implemented in `C` within the `acados` framework using BLASFEO for the linear algebra.
Let us first regard Figure 1, which allows an overall comparison of the computation times using both schemes and additionally shows its fractionation into external function calls, the solution of linear systems and remaining (mainly linear algebra) operations. The Butcher tableaux used are corresponding to the Gauss-(Legendre) methods [3], [17] for different number of stages s, which provide an accuracy of order $2s$.
The plot gives an idea of the computational speedup of using GNSF-IRK instead of a standard IRK implementation, i.e. the ratio $\frac{\text{CPUtime(standard IRK)}}{\text{CPUtime(GNSF-IRK)}}$. One observes that this speedup is monotonically increasing with the number of stages (from a speedup of 1.1 for $s = 1$ to a speedup of 2.9 for $s = 6$).
Additionally, we note that this is mainly because the computational cost to solve the linear systems is increasing at a lower rate compared to the standard IRK implementation. However, we also observe that there are other linear algebra operations to be carried out within GNSF-IRK, which are mainly matrix-matrix multiplications. These have to be carried out to get the derivatives of the residual function (15) with respect to $x, u, \mathbf{v}$ (17) and to then multiply the sensitivities of the lifted variables with $\mathbf{K}_\mathbf{v}, \mathbf{Z}_\mathbf{v}$. Furthermore, within the sensitivity generation, matrix-matrix multiplications have to transform the sensitivities of the lifted variables to the original space via (23) and realizing the chain rule (24).

We proceed by comparing the overall computational performance of the two schemes presented in Figure 2. Here, the CPU time and the relative error are plotted on a logarithmic scale and both GNSF-IRK and standard IRK are used with $s = 1, \ldots, 6$ and different numbers of integration steps, for which the corresponding points are connected by polygonal lines. Note that the relative error is computed with respect to a reference solution that is obtained using a standard IRK method with high accuracy, $s = 8$, $n_{\text{newton}} = 5$ and $n_{\text{steps}} = 400$. The lower left part of these lines can be regarded as the Pareto-front of the bicriterial optimization problem that one would naturally formulate to choose an integration method for this numerical simulation task.
First, we note that one can easily see which lines of IRK and GNSF-IRK are corresponding to each other, i.e., they use the same number of stages and Butcher tableau. These lines provide the same relative error at different computational costs.
Notice that if we regard only one Newton iteration and initialize both schemes with zeros, the numerical solutions

$$0 = \begin{bmatrix} -3\,\ddot{q}_5\,p_{19}\sin\vartheta + 3\,\ddot{q}_4\,p_{18}\cos\vartheta + \ddot{q}_1\,p_8 + \ddot{q}_1\,p_7 + \ddot{q}_1\,p_{28} + q_1\,p_{27} + \dot{q}_1\,p_{26} + 3\,\ddot{q}_1\,p_{21} - F_{\text{thrust}} \\ p_1\,T_{\text{rot}} + \ddot{q}_2\,p_8 + \ddot{q}_2\,p_7 + \ddot{q}_2\,p_{29} + q_2\,p_{27} + \dot{q}_2\,p_{26} + 3\,\ddot{q}_2\,p_{21} \\ -T_{\text{rot}} + 3\,\ddot{q}_4\,p_{16}\sin\vartheta - 3\,\ddot{q}_5\,p_{17}\cos\vartheta + (q_3 - q_6)\,p_6 + \ddot{q}_3\,p_5 + 3\,\ddot{q}_3\,p_{20} + (\dot{q}_3 - \dot{q}_6)\,p_2 \\ -p_{25}\sin\vartheta\,T_{\text{rot}} + 3\,\ddot{q}_3\,p_{16}\sin\vartheta - F_{\text{thrust}}\,p_{23}\cos\vartheta + 3\,\ddot{q}_1\,p_{18}\cos\vartheta + 3\,\ddot{q}_4\,p_{14} + 3\,q_4\,p_{12} + 3\,\dot{q}_4\,p_{10} \\ p_{24}\cos\vartheta\,T_{\text{rot}} - F_{\text{thrust}}\,p_{22}\sin\vartheta - 3\,\ddot{q}_1\,p_{19}\sin\vartheta - 3\,\ddot{q}_3\,p_{17}\cos\vartheta + 3\,\ddot{q}_5\,p_{15} + 3\,q_5\,p_{13} + 3\,\dot{q}_5\,p_{11} \\ p_9\,T_{\text{gen}} + (\ddot{q}_6\,p_4 + \ddot{q}_6\,p_3)\,{p_9}^2 + (q_6 - q_3)\,p_6 + (\dot{q}_6 - \dot{q}_3)\,p_2 \end{bmatrix} \tag{26}$$

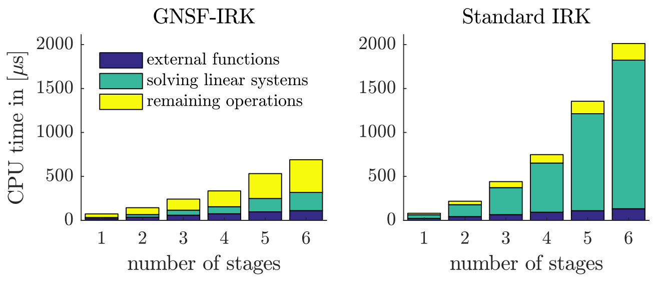

Fig. 1. Timing comparison: standard IRK vs. GNSF-IRK ($n_{\text{newton}} = 3$).

may differ, as the initializations are not equivalent. Numerical experiments in [12] showed that for reasonably transcribed GNSF models, see Section IV, the GNSF-IRK initialization is typically preferable. The reasoning is that this initialization takes linear dependencies into account and assumes the nonlinear term ϕ to be zero.

Again, we observe that the computational speedup of using GNSF-IRK increases when more stages are used. The distance between the lines corresponding to $s = 1$ is relatively small compared to the ones corresponding to $s = 6$. This results in different Pareto-fronts for standard IRK and GNSF-IRK. In the second plot of Figure 2, one observes that the Pareto-front for GNSF-IRK corresponds to $s = 3$ for low accuracy ($\geq 10^{-3}$), followed by $s = 4$ for accuracies in the range of 10^{-5} to 10^{-3}. For more accurate solutions, the plot suggests to use GNSF-IRK with $s = 6$ (or even larger, not included in the plot).

However, if we regard the Pareto-front of the standard IRK, we observe that it consists only of the Butcher tableaux with $s = 3, 4$ considering accuracies lower than 10^{-2}.

This shows that an optimal choice of the integration order is very much dependent on whether a structure exploiting or a standard IRK implementation is used.

In order to arrive at a fair comparison, we filtered out the Pareto optimal points and compared the Pareto fronts. In Figure 3 the Pareto fronts of the GNSF-IRK and the standard IRK scheme are visualized by a stair approximation through the Pareto optimal points. Note that for higher accuracies (i.e. relative error $< 10^{-4}$) the Pareto-front for GNSF-IRK contains only methods of very high order, i.e. 8 to 14. In contrast, the Pareto front of the standard IRK scheme often suggests a method with one stage less than the optimal choice for a given accuracy.

Thus we chose to make a timing comparison by comparing the optimal timings suggested by this approximation of the Pareto fronts for varying accuracies. Note that the speedup of GNSF-IRK compared to the standard IRK is especially good for high accuracies. The maximal speedup in Figure 4 is 2.9. However, for practical applications a speedup greater than 2 is a realistic estimation.

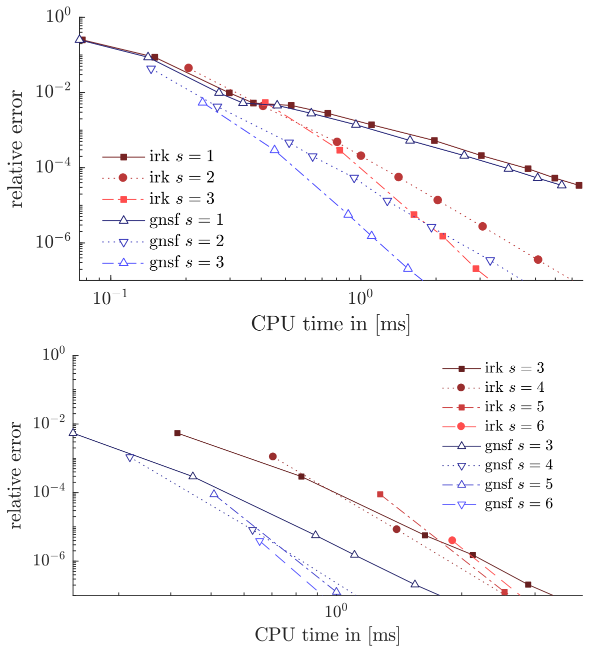

Fig. 2. Standard IRK and GNSF-IRK with different orders for $n_{\text{newton}} = 3$ and varying integration step sizes.

VI. Conclusions & Outlook

In this paper, we extended the concept of exploiting linear dependencies of a continuous time model within an IRK scheme to DAEs. Previous ideas were combined to arrive at a structured dynamic system class that allows one to exploit various types of linearities. The presented scheme has been implemented in the framework of the open-source software `acados`. Furthermore, we presented an automatic transcription method that allows a convenient use of the integrator and is also part of `acados`.

Both the transcription and integration algorithm were applied to a wind turbine model. The experiments showed that a speedup of factor greater than two can be achieved. We observed that using the structure exploiting IRK scheme influences the optimal choice of order and step size, making higher order methods (and larger step sizes) more beneficial.

One could investigate further, how the matrix-matrix multiplications, especially within the sensitivity generation should be realized. Instead of calling BLASFEO routines, one could code generate methods to perform these

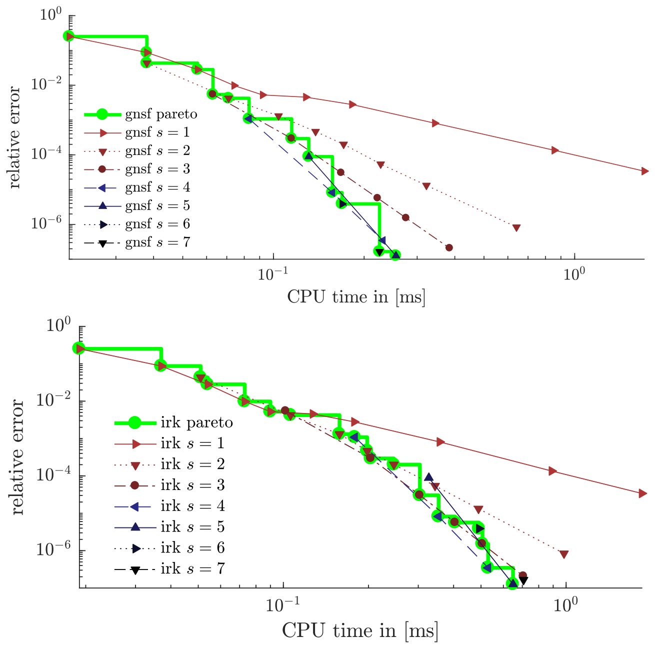

Fig. 3. Pareto front GNSF-IRK and standard IRK

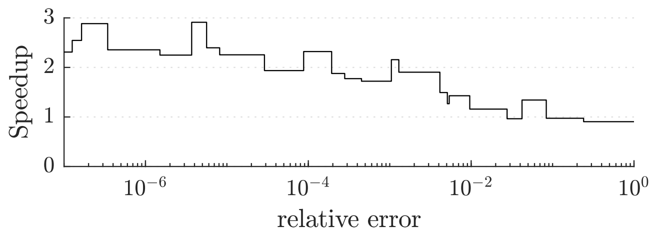

Fig. 4. Speedup comparing the Pareto fronts

multiplications exploiting the sparsity patterns of the matrices. It could be beneficial to generate the gathered residual function (15) directly and omit the linear input matrices $L_x, L_{\dot{x}}, L_z, L_u$. However, both ideas would contradict the modular software framework of `acados`.
Thus, we rather suggest to extend the algorithms from Sections III, IV to a multi-stage GNSF structured dynamic system formulation, following the idea in [17, Fig.4.4]. This should help to obtain bold matrices that are relatively dense also for larger models.

Future extensions include the option to efficiently propagate second order sensitivities within GNSF-IRK and the implementation of a lifted Newton version of the scheme [4]. Additionally, one could investigate how to efficiently combine the scheme in [7] and the one presented here with the IRK structure exploitation [5], [3], [4].

References

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

[2] C. Rao, *Moving Horizon Estimation of Constrained and Nonlinear Systems*. PhD thesis, University of Wisconsin–Madison, 2000.

[3] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics, Berlin: Springer, 2nd ed., 1996.

[4] R. Quirynen, S. Gros, and M. Diehl, "Inexact Newton based lifted implicit integrators for fast nonlinear MPC," in *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*, pp. 32–38, 2015.

[5] J. Butcher, "On the implementation of implicit Runge-Kutta methods," *BIT Numerical Mathematics*, vol. 16, no. 3, pp. 237–240, 1976.

[6] 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.

[7] S. Gros, R. Quirynen, A. Schild, and M. Diehl, "Implicit integrators for linear dynamics coupled to a nonlinear static feedback and application to wind turbine control," in *Proceedings of the IFAC World Congress*, 2017.

[8] "acados." `www.acados.org`, 2018.

[9] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, R. Quirynen, and M. Diehl, "Towards a modular software package for embedded optimization," in *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*, 2018.

[10] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "BLASFEO: Basic linear algebra subroutines for embedded optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 4, pp. 42:1–42:30, 2018.

[11] "BLASFEO." `https://github.com/giaf/blasfeo`, 2016.

[12] J. Frey, "Structure Exploiting Integrators for model predictive control," Master's thesis, University of Freiburg, 2018.

[13] 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*, 2018.

[14] S. Gros, M. Vukov, and M. Diehl, "A real-time MHE and NMPC scheme for wind turbine control," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2013.

[15] S. Gros and A. Schild, "Real-time Economic Nonlinear Model Predictive Control for Wind Turbine Control," *International Journal of Control*, vol. 90, no. 12, pp. 2799–2812, 2017.

[16] R. Mitze, D. Dillkötter, A. S. S. Gros, and M. Mönnigmann, "Fast and smooth surface b-spline interpolation for regularly spaced data used in system modeling to make mpc real-time feasible," in *Proceedings of the European Control Conference (ECC)*, 2018.

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