Master's Thesis

Structure Exploiting Integrators for Model Predictive Control

Jonathan Paul Frey

Albert-Ludwigs-University Freiburg

Faculty of Mathematics and Physics

Department of Mathematics

Chair for System Theory, Control and Optimization

November 07$^{\text{th}}$, 2018

Writing period

07. 05. 2018 – 07. 11. 2018

Examiners

Prof. Dr. Moritz Diehl
Prof. Dr. Patrick Dondl

Advisers

Dimitris Kouzoupis
Dr. Gianluca Frison

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Acknowledgements

I would like to thank all the people who helped me while writing this thesis. First, I would like to thank Prof. Dr. Moritz Diehl for supervising my work and sharing his wide variety of fruitful ideas.
I gratefully acknowledge my supervisors Dimitris Kouzoupis and Dr. Gianluca Frison for their continuous support and feedback through the whole thesis project.
Also, I would like to express my sincere gratitude to Dr. Rien Quirynen, his ideas, expertise in the topic as well as detailed explanations have significantly contributed to this work.
Moreover, I would like to thank all the people in the syscop office for their willingness to share their knowledge and software skills, which made my work in this professional research environment a pleasant experience. Hereby I want to mention additionally Tommaso, Andrea, Katrin, Robin, Tobi, Matilde, Jochem and Misha.
I gratefully acknowledge the cooperation within the eco4wind project, whereby the collaborations with Dr. Axel Schild (IAV GmbH) and Dr. Jens Geisler (Senvion GmbH) should be highlighted. Moreover, I would like to express my gratitude for inspiring discussions with Robin Verschueren and Eugen Nuss in the framework of the DFG research unit FOR 2401.
I also want to give thanks to my parents and sisters, who continuously supported me in an unconditional way over the last years and since I can remember.
Last but not least, I want to thank all my friends, who made the last three years in Freiburg such an enjoyable time I would not want to miss. Hereby, I also want to highlight my Ultimate Frisbee team Disconnection Freiburg, aka Kiel 2, for taking me to a variety of fun tournament and stepping up my game.

Abstract

Optimization based control strategies, such as 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 are still a challenge. The simulation of the model and its sensitivity propagation are key tasks that have to be carried out frequently within most NMPC algorithms. Thus, it is critical for an optimal control software package to have efficient algorithms for these tasks available. A software package for optimal control, that was recently developed by the team of Prof. Moritz Diehl, is `acados`. The development of integration schemes in `acados`, also called integrators, constitutes a major part of this thesis. In the following, the main contributions of this thesis are summarized.

The Implicit Runge-Kutta (IRK) integrator is maintained and extended to support differential-algebraic equations (DAE) and the option to propagate exact second order sensitivities.

The previous work on structure exploitation within IRK methods are presented and compared. A dynamic system structure, which can also treat index-1 DAEs and is called "Generalized Nonlinear Static Feedback" (GNSF), combines the approaches that exploit linear dependencies within the dynamic model. An efficient IRK scheme that exploits the GNSF structure, thus called GNSF-IRK, was derived and implemented in `acados` with the option to efficiently propagate forward and adjoint first order sensitivities.

Additionally, an algorithm is proposed that can automatically transcribe most index-1 DAE systems into the presented GNSF structure, such that the GNSF-IRK scheme can be used conveniently.

Within numerical experiments, both the transcription algorithm and GNSF-IRK are applied to various dynamic models, of which some are correspond to industrial wind turbines. The analysis of the experiments shows how the algorithms should be used and how GNSF-IRK can outperform the standard IRK method.

Zusammenfassung

Nichtlineare modellprädiktive Regelung (NMPC) und Zustandsschätzung auf bewegten Horizonten (MHE) sind bekannte Echtzeitbetriebsführungsstrategien für eine Vielfalt an physikalischen Systemen. Im Bereich der eingebetteten Echtzeitoptimierung sind Systeme mit schneller und nichtlinearer Dynamik jedoch immer noch eine Herausforderung. Die Simulation des Modells und seine Sensitivitätsanalyse sind Kernaufgaben, die im Rahmen der meisten NMPC Algorithmen häufig ausgeführt werden müssen. Für Softwarepakete im Bereich der optimalen und eingebetteten Steuerung ist es daher entscheidend effiziente Algorithmen für derartige Aufgaben bereitzustellen.

Ein solches Softwarepaket für optimale Steuerung, das jüngst im Team von Prof. Moritz Diehl entwickelt wurde, ist `acados`. Die hier vorgestellte Arbeit hat zur Entwicklung der Simulationsalgorithmen in `acados`, auch genannt Integratoren, wesentlich beigetragen. Im Folgenden, werden die wichtigsten Aspekte dieser Arbeit zusammengefasst.

Der implizite Runge-Kutta (IRK) Integrator wurde gepflegt und erweitert, sodass auch die Simulation differentiell-algebraischer Gleichungen (DAE) und die Option der Sensitivitätsanalyse zweiter Ordnung unterstützt wird.

Frühere Arbeiten zu IRK Algorithmen, welche die Struktur der Newton-Matrix nutzen, werden dargestellt und verglichen. Die Struktur eines dynamischen Systems, das auch index-1 DAEs handhaben kann und "Verallgemeinerte Nichtlineare Statische Rückkopplung" (GNSF) genannt wird, wird präsentiert. Diese Struktur kombiniert die Ansätze, lineare Abhängigkeiten der Systemdynamik zu nutzen. Ein effizientes IRK Verfahren, das die GNSF Struktur ausnutzt (GNSF-IRK), wurde hergeleitet und im Rahmen von `acados` implementiert. Der Algorithmus bietet auch die Möglichkeit herkömmliche und adjungierte Sensitivitäten erster Ordnung zu propagieren.

Zusätzlich wird ein Algorithmus vorgestellt, welcher automatisch die meisten index-1 DAE Modelle in die präsentierte GNSF Struktur umschreiben kann, um das GNSF-IRK Verfahren praktikabel nutzen zu können.

In numerischen Experimenten wird der Algorithmus zum Umschreiben des dynamischen Systems und GNSF-IRK an verschiedenen Modellen getestet, von denen einige die Dynamik industrieller Windenergieanlagen modellieren. Die Analyse der Experimente zeigt, wie die Algorithmen genutzt werden sollten und wie GNSF-IRK das gewöhnliche IRK Verfahren bezüglich der Rechenzeit schlagen kann.

Contents

List of Abbreviations and Symbols

Abbreviations

NMPC . . . . Nonlinear Model Predictive Control

MPC . . . . . Model Predictive Control

MS . . . . . Multiple Shooting

NLP . . . . . Nonlinear Programming

OCP . . . . . Optimal Control Problem

SQP . . . . . Sequential Quadratic Programming

ODE . . . . . Ordinary Differential Equations

DAE . . . . . Differential Algebraic Equation

IVP . . . . . Initial Value Problem

RK . . . . . Runge-Kutta

IRK . . . . . Implicit Runge-Kutta

ERK . . . . . Explicit Runge-Kutta

AD . . . . . Algorithmic (or Automatic) Differentiation

FD . . . . . Finite Differences

IND . . . . . Internal Numerical Differentiation

END . . . . . External Numerical Differentiation

VDE . . . . . Variational Differential Equations

CAS . . . . . Computer Algebra System

NSF . . . . . Nonlinear Static Feedback

NSFLO . . . . Nonlinear Static Feedback coupled to a Linear Output System

GNSF . . . . . Generalized Nonlinear Static Feedback

Symbols and Notation

$\mathbb{0}$	zero matrix
$\mathbb{1}$	unit matrix
$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
$\mathbb{C}$	the set of complex numbers
$\mathbb{C}^-$	the set of complex numbers with negative real part
$\forall$	for all
$\exists$	there exists
$\otimes$	Kronecker product
$^\top$	transpose
$[\ ]$	empty matrix
$[B_{ij}]_{i,j=1,\ldots,n}$	matrix consisting of blocks $B_{i,j}$ where i and j denote the block row and column index respectively
$\operatorname{diag}(B_i)_{i=1,\ldots,n}$	block diagonal matrix containing the blocks B_i on its diagonal
$A_{I,J}$	submatrix of A consisting of all rows with indices $i \in I$ and columns $j \in J$. If I (respectively J) is just an element only the corresponding row (column) is meant. Writing ":" instead of I (respectively J) means all rows (columns) are taken.
$\mathbb{Z}_n^m$	integers i with $n \leq i \leq m$
$x(\cdot)$	the differential states
$\dot{x}(\cdot)$	the differential state derivatives
$u(\cdot)$	the control inputs
$z(\cdot)$	the algebraic states
s	number of stages in a RK scheme
n_{steps}	number of steps in a RK scheme

1 Introduction

Recently, the team led by Prof. Moritz Diehl developed a modular software package for embedded optimization named `acados` [1; 2], which gathers a variety of numerical algorithms for Nonlinear Model Predictive Control (NMPC) that are implemented in `C` using the efficient basic linear algebra library BLASFEO [3; 4] such that it is high performing and can be used on embedded hardware.
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 derivatives of the simulation result with respect to the initial state and control, are key tasks that often have to be carried out 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*.
The work described in this thesis deals with the development of such efficient and structure exploiting integrators for `acados`.

1.1 Real-Time Optimization

Optimization based control techniques like Model Predictive Control (MPC) and Moving Horizon Estimation (MHE) enjoy a growing popularity as their application has become real-time feasible for a growing variety of applications. In this section, we briefly introduce the formulation and solution of real-time estimation and optimal control problems presented in [5; 6].
Figure 1 shows the concept of estimation-based feedback control and how the basic components, controller, estimator and the real system, also referred to as *plant* [7], interact with each other. In this framework, some variables of the system are measured with a certain sample frequency. Using these measurements and the applied control, the estimator (e.g. MHE) determines an estimate of the current state of the system and possibly some parameters. Given these estimates, the constraint and objective functions together with a dynamic model, the NMPC controller calculates the next values of the "optimal" control input, which is then applied to the system.

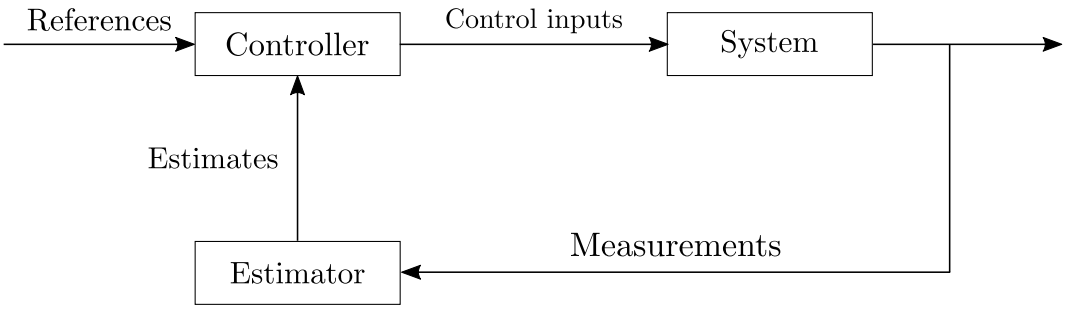

Figure 1: Illustration of the Estimation-based Feedback Control Framework. It is shown how the controller, the estimator and the system interact with each other.

First used in chemical engineering, real-time estimation and control algorithms become evermore popular [8]. Due to increasingly sophisticated hardware and embedded optimization software, sampling times have been reduced from several minutes to microseconds [9] making it feasible for fast changing systems [10]. Moreover, optimization based control is very flexible as one can easily change the problem functions (objective and constraints) and deal with disturbances.

1.1.1 Nonlinear Model Predictive Control

As for both techniques MPC and MHE, a model is needed to describe the system of interest, we assume the following controlled implicit Ordinary Differential Equation (ODE) system to model our system on the horizon $[0, T]$:

$$f(t, \dot{x}(t), x(t), u(t)) = 0, \quad t \in [0, T]\,. \tag{1.1}$$

Hereby, $t \in \mathbb{R}$ denotes the continuous time, $u : \mathbb{R} \to \mathbb{R}^{n_\mathrm{u}}$ the controls, $x : \mathbb{R} \to \mathbb{R}^{n_\mathrm{x}}$ the differential states and $\dot{x}(t) = \frac{\mathrm{d}x}{\mathrm{d}t}$ their time-derivative. It is assumed that the corresponding initial value problems (IVP) have a unique solution. For sufficient conditions on f for the existence and uniqueness of the solution, we refer to the famous Picard-Lindelöf Theorem [11; 12].
Within MPC, different instances of an optimal control problem (OCP) have to be solved repeatedly. The continuous time formulation of a general OCP can be written

as follows [7; 12],

$$\begin{aligned}
\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & \int_0^T L(t,x(t),u(t))\,\mathrm{d}t + M(x(T)) && \text{(1.2a)}\\
\text{subject to} \quad & 0 = x(0) - \bar{x}_0, && \text{(1.2b)}\\
& 0 = f(t,x(t),\dot{x}(t),u(t)), && \text{(1.2c)}\\
& 0 = g(t,x(t),u(t)), && \text{(1.2d)}\\
& 0 \geq h(t,x(t),u(t)) \quad \text{for all } t \in [0,T]. && \text{(1.2e)}
\end{aligned}$$

The objective, (1.2a) is built as a combination of the way-cost, or Lagrange term, $L(\cdot)$ and the end-cost, or Mayer term, $M(\cdot)$ [13]. Additionally, we introduced the path constraints using $g(\cdot)$ for the equalities and $h(\cdot)$ for the inequalities.
Note that in many applications both $L(\cdot)$ and $M(\cdot)$ are least-square functions because they are used within tracking formulations. This property is very handy as specialized methods can be applied, e.g. Gauss-Newton Hessian approximations can be used efficiently [14].
The MPC approach can now be described by the following steps [12]:

1. Get estimate of current state $\bar{x}_0$ and possibly some parameters from the estimator.

2. Solve the OCP (1.2) approximately. This has to be done fast and in deterministic runtime, since the sampling time T_s is a strict upper bound.

3. Apply the optimal control solution $u(t), t \in [0,\, T_\mathrm{s}]$.

4. Repeat after a fixed sampling time T_s.

In the following, the direct time dependency of the functions L, f, g, h will be omitted. Firstly, because they are rather rare in the context of MPC, and secondly because models with time dependency can be reformulated by introducing an additional "clock state" [14].

1.1.2 Moving Horizon Estimation

The optimization problem which has to be solved repeatedly in MHE is very similar to the OCP (1.2). However, within MHE the controls $u(t)$ are known as they are provided to the estimator, see Figure 1. For the dynamic system used in the MHE optimization problem, disturbances $w(t)$ are introduced to represent the plant-model

mismatch, resulting in the following ODE formulation

$$0 = f(t, x(t), \dot{x}(t), u(t), w(t)). \tag{1.3}$$

The OCP that needs to be solved in MHE is based on the measurements that are forwarded to the estimator and could be formulated as

$$\begin{aligned}
\underset{x(\cdot), w(\cdot)}{\text{minimize}} \quad & \int_0^{T_\mathrm{e}} L(t, x(t), u(t), w(t))\,\mathrm{d}t + M(x(T_\mathrm{e})) && \text{(1.4a)}\\
\text{subject to} \quad & 0 = f(t, x(t), \dot{x}(t), u(t), w(t)), && \text{(1.4b)}\\
& 0 = g(t, x(t), u(t), w(t)), && \text{(1.4c)}\\
& 0 \geq h(t, x(t), u(t), w(t)) \quad \text{for all } t \in [0, T_\mathrm{e}]. && \text{(1.4d)}
\end{aligned}$$

The estimation is then given by $x(T_\mathrm{e})$. The main differences in comparison to the OCP (1.2) are that firstly $u(t)$ is given and the disturbances $w(t)$ can be seen as inputs and secondly that the initial state $x(0)$ is free [12; 7].
Note that all the presented OCPs are still of infinite dimension. In the following, we will only regard the OCP (1.2) to explain how an approximation of the optimal solution can be computed. Thus, the continuous time problem will be discretized, such that it can be tackled with numerical methods.

1.1.3 Optimal Control Algorithms

In optimal control, most algorithms are based on direct methods, in which first the OCP (1.2) is discretized and then a Newton-type optimization method is applied to get an approximation of the optimal solution.
The alternative is the indirect approach. Here, first differential equations are derived to describe the optimal solution via Pontryagins Maximum Principle (PMP) and then these equations are solved in discrete time [15; 14]. In this thesis, only the direct approach is considered further.
For the discretization, an appropriate parametrization of the control inputs $u(t)$ is needed. Most often, a piecewise constant control parametrization is chosen, but there are more general parametrizations, like piecewise polynomials of some fixed degree. Let us assume that the time horizon $[0, T]$ is discretized into N shooting intervals, parameterizing the functions $x(\cdot), u(\cdot)$ as discrete variables $x_0, x_1, \ldots, x_N$ and $u_0, \ldots, u_{N-1}$ respectively.
There are two ways to obtain the finite-dimensional optimization problem, the sequential and the simultaneous approach.
In the *sequential approach*, also called *single shooting*, the system simulation and

the optimization are performed sequentially in each iteration. Therefore, only $x_0, u_0, \ldots, u_{N-1}$ are kept as optimization variables, whereas $x_1, \ldots, x_N$ are eliminated using the fact that the state trajectory of the solution is given by the initial state x_0 and the control inputs. For the simulation part, one can use integrators, as presented later in this thesis.

In contrast to the above, using the *simultaneous approach*, the whole parametrization $x_0, \ldots, x_N, u_0, \ldots, u_{N-1}$ is kept as optimization variables and the simulation and optimization tasks are performed simultaneously in one Newton-type iteration. This can be done using either *Multiple Shooting* (MS) or a *direct transcription* method, of which the most popular subclass is the family of direct collocation methods [14; 16]. The simultaneous approach leads to an optimization problem which has much more variables and inherently a very specific structure, which involves a huge potential for exploitation [17]. Both approaches result in very different convergence properties in the Newton-type optimization. Especially for unstable or highly nonlinear systems, faster local convergence rates are observed when using the simultaneous approach. Intuitively, this is because the nonlinearity of the dynamics is equally distributed over the horizon [18].

Using the discretization we introduced above, the following discrete MS formulation can be derived from the continuous time OCP (1.2):

$$\underset{\substack{x_0,\ldots,x_N\in\mathbb{R}^{n_\mathrm{x}},\\ u_0,\ldots,u_{N-1}\in\mathbb{R}^{n_\mathrm{u}}}}{\text{minimize}} \quad \sum_{i=0}^{N-1} l_i(x_i, u_i) + M(x_N) \tag{1.5a}$$

$$\text{subject to} \quad \bar{x}_0 - x_0 = 0, \tag{1.5b}$$

$$x_{i+1} - \psi\left(x_i, u_i\right) = 0, \quad i = 0, \ldots, N-1, \tag{1.5c}$$

$$g_i(x_i, u_i) = 0, \quad i = 0, \ldots, N-1, \tag{1.5d}$$

$$h_i(x_i, u_i) \leq 0, \quad i = 0, \ldots, N-1. \tag{1.5e}$$

Note that (1.5d), (1.5e) are relaxations of the corresponding constraints (1.2d) and (1.2e), as they are only enforced on the shooting nodes. In practice, this often limits the constraint violations sufficiently, when a relatively fine discretization is used [12]. Note that the equations (1.5c) refer to the simulation of the dynamic model in (1.1), which is the task of an integrator.

In the MS formulation, any integrator could be used, including variable step and variable order methods. However, for real-time applications, one has fix the code for the integration over one shooting interval [19].

Including the equations of this integrator in the problem, a direct transcription

formulation of the OCP is obtained:

$$\begin{aligned}
\underset{\substack{x_0,\dots,x_N\in\mathbb{R}^{n_\mathrm{x}},\\ u_0,\dots,u_{N-1}\in\mathbb{R}^{n_\mathrm{u}}}}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \tilde{l}_i(x_i,u_i) + \tilde{M}(x_N) && \text{(1.6a)}\\
\text{subject to} \quad & \bar{x}_0 - x_0 = 0, && \text{(1.6b)}\\
& x_{i+1} - \tilde{\psi}_i\left(x_i, z_i, u_i\right) = 0, \quad i = 0,\dots,N-1, && \text{(1.6c)}\\
& \tilde{g}_i(x_i,z_i,u_i) = 0, \quad i = 0,\dots,N-1, && \text{(1.6d)}\\
& \tilde{h}_i(x_i,z_i,u_i) \leq 0, \quad i = 0,\dots,N-1. && \text{(1.6e)}
\end{aligned}$$

This is a fully discretized OCP whereby $z_i \in \mathbb{R}^{n_z}$ denote the integration variables and algebraic states. The fixed step integrator used to simulate the system from time point k to $k+1$ is denoted as $\tilde{\psi}_k$ and its internal variables are gathered in z_k. The function $\tilde{g}_i$ gathers the former equality constraints (1.5d) and the integration equations. Hereby, we assume that the integration variables z_i are uniquely defined by the values x_i and u_i.
Note that $\tilde{\psi}_i$ represents a very general single-step integration scheme, as it could be any RK method for a either an ODE or a DAE system. For the sake of completeness a short introduction on Newton-type optimization is given in the next subsection.

1.2 Nonlinear Optimization and Newton's Method

Nonlinear Optimization This section aims to give a brief introduction into the terminology of nonlinear optimization.
Typically, within the field of nonlinear optimization, one regards a *Nonlinear Programming (NLP)* formulation which has the following form

$$\begin{aligned}
\underset{x\,\in\,\mathbb{R}^n}{\text{minimize}} \quad & F(x)\\
\text{subject to} \quad & g(x) = 0,\\
& h(x) \leq 0,
\end{aligned} \tag{1.7}$$

where $F : \mathbb{R}^n \to \mathbb{R}$ is the objective function and $g : \mathbb{R}^n \to \mathbb{R}^{n_\mathrm{g}}, h : \mathbb{R}^n \to \mathbb{R}^{n_\mathrm{h}}$ are the equality and inequality constraints respectively. All problem functions, F, g, h are assumed to be at least once continuously differentiable [14; 20]. Note that the optimization problems in (1.6) and (1.5) are just structured NLPs.
The first order necessary conditions for the general NLP formulation (1.7) have been derived by Karush, Kuhn and Tucker and are thus called *KKT conditions* [21].

Definition 1.1 (KKT point) The triple (x^*, λ^*, μ^*) with $\lambda^* \in \mathbb{R}^{n_\mathrm{g}}, \mu^* \in \mathbb{R}^{n_\mathrm{h}}$, is a *KKT point* of the NLP (1.7) if the following *KKT conditions* hold

$$\nabla F(x^*) + \nabla g(x^*)\lambda^* + \nabla h(x^*)\mu^* = 0 \tag{1.8a}$$
$$g(x^*) = 0 \tag{1.8b}$$
$$h(x^*) \leq 0 \tag{1.8c}$$
$$\mu_i^* \geq 0 \quad \forall i = 1, \ldots, n_\mathrm{h} \tag{1.8d}$$
$$\mu_i^* h_i(x^*) = 0 \quad \forall i = 1, \ldots, n_\mathrm{h}. \tag{1.8e}$$

Under some mild conditions on the constraints, thus called constraint qualifications, one can state that for every local minimizer x^* of NLP (1.7), there exist multipliers λ^*, μ^*, such that (x^*, λ^*, μ^*) is a KKT point.

Newton-type Methods Newton's method (also Newton-Raphson method) is one of the most famous numerical methods [22]. To clarify the terminology, we want to briefly point out the difference between Newton-type optimization methods and Newton-type iterations to tackle root finding problems [23].
A *Newton-type optimization* method tries to find a KKT point by linearizing the problem functions and performing iterations similar to a classical Newton's method. Newton-type iterations for root finding problems are just variants of the classical Newton's method, in which for example an approximation of the Hessian matrix is taken instead of the exact one. They could be used within an IRK scheme to solve the system of nonlinear equations. However, in this work, we restrict ourselves to using the classical full-step Newtown's method within IRK schemes.

1.3 Implementation and Software

This section aims at giving a brief overview on the software frameworks used within this thesis project.

1.3.1 `acados`

`acados` is a recently developed modular open-source software package for embedded optimization [1; 2]. It gathers a variety of numerical algorithms for NMPC that are implemented in `C` using the efficient basic linear algebra library BLASFEO, see Section 1.3.2.
In some sense, `acados` aims to replace the `ACADO Toolkit` which was widely used and previously developed within the team led by Prof. Moritz Diehl [24; 18; 25; 26].

The main paradigm of the ACADO Toolkit is code generation, which comes with a high price to pay in flexibility, maintainability and extensibility of the software. The new software package acados drops the code generation pradigm and aims to combine the principles of flexibility, reproducibility, modularity and efficiency. The algorithmic components in the C part of acados, such as integrators, come with a generic C interface, through which any dynamic model, cost and constraint function, etc. can be used. In principle, this code could be provided by any modeling tool with the necessary differentiation and code generation capabilities, but typically, this is realized by code generation in CasADi (presented in Section 1.3.3).

1.3.2 BLASFEO

BLASFEO stands for "Basic Linear Algebra Subroutines For Embedded Optimization" and is a dense linear algebra library providing high-performance implementations of BLAS- and LAPACK-like routines. Unlike other linear algebra libraries, it aims at computational performance for small to medium matrices, i.e. for sizes up to a few hundred, which typically occur within MPC [3; 4].
All implicit integrators in acados are implemented using BLASFEO to perform the linear algebra operations. This ensures also the comparability of the results presented in Chapter 5.

1.3.3 CasADi

The open-source software CasADi is a general-purpose numerical optimization framework with a focus on optimal control. CasADi started as a tool for Algorithmic Differentiation (AD) using a syntax similar to Computer Algebra Systems (CAS), which explains the name. It is written in C++ and comes with an interface to Python, MATLAB and Octave [27; 28].
Important CasADi features, that are involved in the current acados workflow, are the AD and C Code generation capabilities. The CasADi algorithms for Jacobian and Hessian generation first calculate the sparsity pattern of the matrix of interest and subsequently use a heuristic to find a minimal set of directional derivatives, from which the full Jacobian can be constructed. Both are done using graph coloring techniques [29].

2 Dynamic Systems, Integration Methods and Sensitivity Propagation

This chapter is organized as follows. First, Section 2.1 gives a brief introduction into the theory of dynamic systems. In Section 2.2, we introduce numerical integration methods, which can be used to approximate trajectories of a dynamic system. Here, the main focus are Runge-Kutta methods. Finally, Section 2.3 presents how these integration methods can be extended to additionally compute the sensitivities of the simulation result.

2.1 Controlled Dynamic Systems

In order to stay in the context of control, we want to introduce *controlled dynamic systems*, i.e. systems that can be manipulated externally, [12].

2.1.1 Ordinary Differential Equations

Definition 2.1 We regard the time $t \in \mathbb{R}$, the differential states $x(t) \in \mathbb{R}^{n_\mathrm{x}}$ and control inputs $u(t) \in \mathbb{R}^{n_\mathrm{u}}$. A system of *Ordinary Differential Equations* (ODE) describes the dynamic evolution of the state vector x and can be written as

$$0 = f^{\mathrm{impl}}(\dot{x}(t), x(t), u(t)). \tag{2.1}$$

Here, the notation $\dot{x}(t) = \frac{\partial x(t)}{\partial t}$ is used for the differential state derivatives and the Jacobian matrix $\frac{\partial f^{\mathrm{impl}}}{\partial \dot{x}}$ is assumed to be invertible, $\forall t$.
We will sometimes regard the special case of an *explicit* ODE, which reads as

$$\dot{x}(t) = f^{\mathrm{expl}}(x(t), u(t)) \tag{2.2}$$

We will often omit the time dependency and write $f^{\mathrm{impl}}(\dot{x}, x, u) = 0$, because in the context of optimal control there is typically no time dependency and it simplifies the notation.

2.1.2 Differential-Algebraic Equations

Some more complex physical systems can not easily be described by an ODE system. Instead, they can be modeled naturally by a system of differential-algebraic equations. A very popular example is the class of multibody systems, which are typically modeled

by applying Lagrange mechanics on each of the present bodies and linking them by algebraic constraints [30].

Definition 2.2 We introduce the differential states $x(t) \in \mathbb{R}^{n_\mathrm{x}}$, control inputs $u(t) \in \mathbb{R}^{n_\mathrm{u}}$ and the algebraic variables $z(t) \in \mathbb{R}^{n_\mathrm{z}}$ for time $t \in \mathbb{R}$ to define the following set of (fully-implicit) Differential-Algebraic Equations (DAE) to describe the behavior of a dynamic system

$$0 = f^{\mathrm{impl}}(\dot{x}(t), x(t), z(t), u(t)). \tag{2.3}$$

The DAE is said to be of index 1, if the Jacobian matrix $\frac{\partial f^{\mathrm{impl}}}{\partial \dot{x}, z}$ is invertible.

The following definition, adapted from [31], introduces the differential index of a DAE, which is often referred to just as "index of a DAE" [32].

Definition 2.3 The differential index of the DAE system (2.3) is defined as the minimal $i \in \mathbb{N}$ such that the following system is an ODE

$$\frac{\partial^j}{\partial t^j} f^{\mathrm{impl}}(\dot{x}(t), x(t), z(t), u(t)) = 0, \quad \text{for } j = 0, \ldots, i \tag{2.4}$$

In contrast to the previous definition, one often regards a semi-explicit DAE, which is more structured and can be written as

$$\dot{x}(t) = f^{\mathrm{expl}}(x(t), z(t), u(t)) \tag{2.5a}$$

$$0 = g(x(t), z(t), u(t)) \tag{2.5b}$$

where the function $g(\cdot)$ defines the algebraic equations. A semi-explicit DAE is of index 1, if the Jacobian $\frac{\partial g}{\partial z}(\cdot)$ is invertible. It should be mentioned that the structure of a semi-explicit DAE can be exploited within integration schemes, as proposed in [19, Sec. 4.1.3].
In the following part of this thesis, we will always assume DAE systems to be of index 1. In practice, one usually reformulates high-index DAE systems until the differential index is reduced to 1 or 0 by applying index reductions techniques, as described in [33]. Therefore, the loss of generality here is rather small.

Definition 2.4 An index-1 DAE or ODE in combination with an initial condition $x(t_0) = x_0$ and a given control trajectory $u(t)$ on some time interval $t \in [t_0, T]$ is called an *initial value problem* (IVP)

$$0 = f^{\mathrm{impl}}(\dot{x}(t), x(t), z(t), u(t)), \quad x(t_0) = x_0. \tag{2.6}$$

Remark 2.5 (Stiffness) In modern MPC applications, we often have to deal with stiff dynamic models. These kind of models often occur when considering chemical, biological, mechanical and electrical processes [34]. As there is no standardized definition of stiffness, we cite the characterization of Hairer and Wanner, because it is brief and states the most important property regarding numerical simulations: "Stiff equations are problems for which explicit methods don't work" [31]. This is strongly connected to stability properties of numerical simulation methods, see Section 2.2.4.

2.2 Numerical Simulation

In this section, we treat the numerical approximation of the solution of an IVP (2.6).

2.2.1 General Linear Integration Methods

Numerical integration methods can be used to simulate a dynamic system forward in time. Let us now regard the following time-dependent IVP

$$0 = f^{\text{impl}}(t, \dot{x}(t), x(t), u(t)), \quad x(t_n) = x_n. \tag{2.7}$$

The time-dependent formulation helps to understand which values are used to approximate which trajectory at which point in time.
A numerical integration method provides an approximation x_{n+1} of $x(t_{n+1})$, where $x(\cdot)$ denotes the solution of the IVP (2.7).
One distinguishing characteristic is whether the method uses only the numerical solution at the previous point x_n, or a set of previously computed values $x_n, x_{n-1}, \ldots$ and their function values. We refer to these kinds of methods as *Single-step* (also one-step [35]) and *Multistep methods* respectively.
In order to have a general formulation available, we introduce the following flexible notation for single-step methods

$$0 = G(x_n, K_n, u_n), \tag{2.8a}$$

$$x_{n+1} = \Psi(x_n, K_n, u_n). \tag{2.8b}$$

Within the scheme, first the *integration equations* gathered in G are solved (up to some precision) for the *integration variables* K_n. Subsequently the simulation result is obtained by an evaluation Ψ. In this thesis, we nearly exclusively deal with the most important subclass of single-step methods, the Runge-Kutta methods, presented in the next subsection.

2.2.2 Runge-Kutta Methods

We regard Runge-Kutta (RK) methods, which use some function evaluations at intermediate stage points in the interval $[t_n, t_{n+1}]$ to define a numerical solution of the IVP (2.7) at the end of this interval. We use the following general formulation of a *s-stage Runge-Kutta* method

$$0 = f^{\text{impl}}(t_{n,i}, k_i, x_n + T_{\text{int}} \sum_{j=1}^{s} a_{ij} k_j, u_n) \qquad \text{for } i = 1, \ldots, s, \tag{2.9a}$$

$$x_{n+1} = \Psi(x_n, K_n, u_n) = x_n + T_{\text{int}} \sum_{j=1}^{s} b_j k_j, \tag{2.9b}$$

where we denote $t_{n,i} = t_n + c_i T_{\text{int}}$ and use the coefficients a_{ij}, b_j, c_i for $i, j = 1, \ldots, s$, which fully characterize a RK method. These coefficient are typically neatly arranged in a *Butcher tableau*, which reads as:

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \ldots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \ldots & a_{ss} \\ \hline & b_1 & \ldots & b_s \end{array} \tag{2.10}$$

Equation (2.9) is called the *differential* formulation of an RK method, as k_i are numerical approximations of $\dot{x}(t_{n,i})$. In general, we first have to solve the implicit Equation (2.9a) for k_i and subsequently get the simulation result as a weighted sum of the k_i and x_n. In the case of an explicit ODE, an RK method reads as:

$$k_i = f^{\text{expl}}(t_{n,i}, x_n + T_{\text{int}} \sum_{j=1}^{s} a_{ij} k_j, u_n) \quad \text{for } i = 1, \ldots, s \tag{2.11a}$$

$$x_{n+1} = x_n + T_{\text{int}} \sum_{j=1}^{s} b_j k_j \tag{2.11b}$$

Observe that in case of an RK method with strictly lower triangular matrix A, i.e. $a_{ij} = 0$ for all $j \geq i$, we can obtain the solution k_i explicitly, successively for $i = 1, \ldots, s$. For this reason, RK methods with the property that matrix A is lower triangular are called *explicit* RK methods (ERK), schemes without this property are called implicit (IRK). The first introduced (1768) and most simple ERK scheme is the explicit Euler method, which reads as

$$x_{n+1} = x_n + T_{\text{int}} f^{\text{expl}}(t_n, x_n, u_n).$$

The Butcher tableau of this and other famous ERK schemes can be found in Table 1.

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \qquad \begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} \qquad \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ \frac{1}{2} & 0 & \frac{1}{2} & \\ 1 & 0 & 0 & 1 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Table 1: Butcher tableaux of some famous ERK schemes: Explicit Euler, Midpoint Rule, *the* Runge-Kutta method of order 4 (ERK4), from left to right.

2.2.3 Implicit Runge-Kutta Methods

Although for explicit ODEs the integration equations of an ERK method (2.11) are very cheap to solve, it is often recommended to use an IRK method instead. In this general case, Equation (2.9a) is a system of $s\,n_\mathrm{x}$ nonlinear equations and variables, which is typically solved by a Newton-type method, see Section 1.2.
The main benefits of IRK over ERK methods are:

- **Higher order:** There always exists an implicit method with s stages that is of order $2s$, while there is no ERK method with s stages that has an order larger than s [19, p.15]. Moreover, there exists no ERK method with $s \geq 5$ stages that has the same order s, which is one reason why the ERK4 formula from Table 1 is so popular.
- **Better Stability Properties:** These are especially important for stiff problems, a brief discussion is given in the next section.

2.2.4 Stability

In this section, a brief introduction on the stability concepts A- and L-stability is given. Both of them are defined via the test problem $\dot{x}(t) = \lambda x(t)$ with $x(0) = x_0$, often referred to as Dahlquist's equation [36]. The corresponding exact solution is $x(t) = x_0\, e^{\lambda t}$. Applying an RK method to this test problem and solving the corresponding integration equations, we can calculate the factor between the initial state x_n and the simulation result x_{n+1} as an explicit function R, i.e.

$$x_{n+1} = R(T_\mathrm{int}\lambda)x_n = R(z)x_n, \tag{2.12}$$

where the shorthand notation $z = T_{\text{int}}\lambda$ is used. The *region of absolute stability* is defined as

$$\{\, z \in \mathbb{C} \mid |\text{Re}(z)| < 1 \,\}. \tag{2.13}$$

In the context of Dahlquist's test problem, this is the set of values z, for which the numerical solution converges to zero. The exact solution of the test problem converges to zero for all $\lambda \in \mathbb{C}^-$ and is therefore stable. The numerical solution should have the same property which motivates the following definitions.

Definition 2.6 A numerical integration method is called *A-stable*, if its region of absolute stability contains the left complex half-plane $\mathbb{C}^-$. Furthermore, a method is called *L-stable*, if it is A-stable and for its stability function R the condition $\lim_{z\to-\infty} |R(z)| = 0$ holds. It is important to use L-stable methods for the simulation of stiff models, as stiff components are damped out faster [31].

The stability properties of collocation methods are mentioned in Section 2.2.5.

2.2.5 Collocation Methods

An intuitive idea for a numerical integration method is to approximate the solution of the IVP (2.7) by a polynomial. In order to define a collocation method, let us regard s distinct collocation points $0 \leq c_1 < c_2 < \cdots \leq c_s \leq 1$. We require the *collocation polynomial* to be of degree s and to fulfill the $s+1$ conditions

$$p(t_n) = x_n, \tag{2.14}$$

$$0 = f^{\text{impl}}(t_{n,i}, \dot{p}(t_{n,i}), p(t_{n,i}), u(t_{n,i})), \tag{2.15}$$

at the time points $t_{n,i} = t_n + c_i T_{\text{int}}$ for $i = 1, \ldots, s$, since we know these hold for the exact solution $x(\cdot)$.
The corresponding collocation integration method is defined by returning $p(t_{n+1})$, the evaluation of the collocation polynomial at the desired point of time $t_{n+1} = t_n + T_{\text{int}}$. This concept is visualized in Figure 2.
A collocation method can be shown to be equivalent to an IRK method with a Butcher tableau fully defined by the collocation points c_i [35]. The remainder values of the Butcher tableau can be obtained by

$$a_{ij} = \int_0^{c_i} \ell_j(\tau)\mathrm{d}\tau, \quad b_i = \int_0^1 \ell_j(\tau)\mathrm{d}\tau, \quad \forall\, i,j = 1,\ldots,s, \tag{2.16}$$

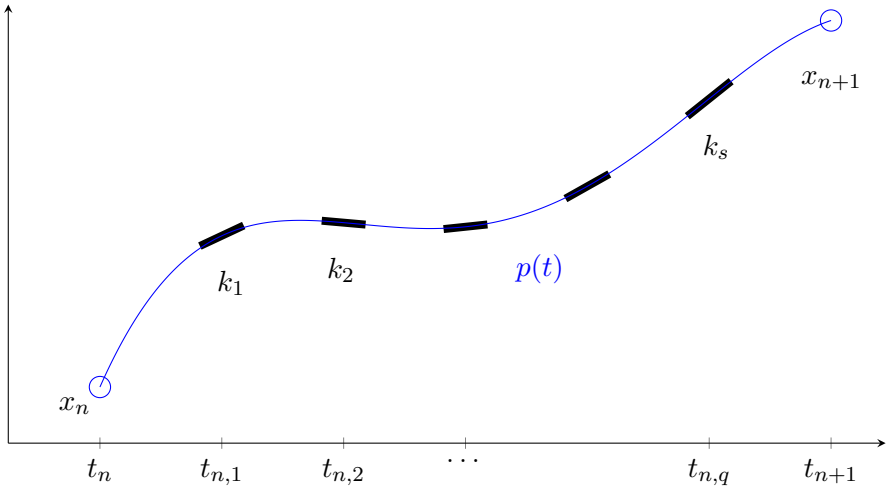

Figure 2: An illustration of the collocation principle. The Runge-Kutta integration variables k_i correspond to the derivative of the approximating polynomial at collocation times $t_{n,i}$. The output of the collocation method is the value of the polynomial at the final time.

whereby the Lagrange polynomials $\ell_j(\cdot)$ are given by

$$\ell_j(t) = \prod_{\substack{k=1 \\ k \neq j}}^{s} \frac{t - c_k}{c_j - c_k}.$$

Since we have seen that a collocation method is fully defined by the choice of collocation points c_i, two optimal ways to choose them are presented in the following paragraphs.

The Gauss-Legendre collocation method In this class of collocation methods the nodes c_i are chosen as the zeros of shifted Legendre polynomials of degree s. It can be shown that the s-stage Gauss-Legendre method is of order $2s$ and A-stable [31, p.72]. Moreover, it can be shown that an A-stable s-stage RK method is at most of order $2s$ [31, Theorem 4.4]. Note that the order result refers to the endpoint t_{n+1}. For the internal stages $p(t_{n,i})$ it is shown that they have an order of at least s [20, p.313]. The first three sets of Gauss-Legendre nodes can be found in Table 2.

The Gauss-Radau IIA collocation method Another famous type of collocation methods are the (Gauss-)Radau IIA methods, which are characterized by choosing

the nodes $c_1, \ldots, c_s$ to be the zeros of following polynomial

$$\frac{\partial^{s-1}}{\partial z^{s-1}} \left(z^{s-1}(z-1)^s \right). \tag{2.17}$$

The s-stage Radau IIA method is of order $2s-1$ [31, p.73] and L-stable [12], which is a desirable property for stiff problems. Moreover, the end point of the simulation interval is always part of the collocation nodes, which is beneficial within the implementation. The first three sets of Radau IIA nodes can be found in Table 2.

s	Gauss-Legendre nodes			(Gauss-)Radau IIA nodes		
1		$\frac{1}{2}$			1	
2	$\frac{1}{2} - \frac{\sqrt{3}}{6}$		$\frac{1}{2} + \frac{\sqrt{3}}{6}$	$\frac{1}{3}$		1
3	$\frac{1}{2} - \frac{\sqrt{15}}{10}$	$\frac{1}{2}$	$\frac{1}{2} + \frac{\sqrt{15}}{10}$	$\frac{4-\sqrt{6}}{10}$	$\frac{4+\sqrt{6}}{10}$	1

Table 2: The first three sets of collocation points of the Gauss-Legendre and Radau IIA methods respectively

2.2.6 IRK Methods for DAEs

Runge-Kutta methods can be generalized for the case of DAE systems in a straightforward way. In addition to the integration variables $k_1, \ldots, k_s$, which we had for ODEs, we introduce the integration variables $z_1, \ldots, z_s \in \mathbb{R}^{n_z}$, which are approximations of the algebraic states at the stage points. Extending RK methods to an index-1 DAE of the form (2.3), the following system of integration equations has to be solved

$$0 = f^{\text{impl}} \left(k_i, x_n + T_{\text{int}} \sum_{j=1}^{s} a_{ij} k_j, z_i, u \right) \quad \forall\, i = 1, \ldots, s \tag{2.18a}$$

$$x_{n+1} = x_n + T_{\text{int}} \sum_{j=1}^{s} b_j k_j. \tag{2.18b}$$

Typically, the nonlinear system consisting of (2.18a) is solved by performing Newton-type iterations as in the ODE case (2.9). The expression for x_{n+1} is the same as in the ODE case.

Often the integrator should also report an approximation of the algebraic variables z. However, there are different possibilities on how the z-values are reported. The following aims to give a brief overview on that:

- **Where to approximate z:** Classically, one reports a consistent value pair (x_{n+1}, z_{n+1}), see e.g. [37, p.182]. However, in the context of embedded optimization, the values of the algebraic variables are mostly used to evaluate constraint or objective functions. In the discretization of an OCP, the constraints over one shooting interval are typically discretized by one evaluation of the corresponding function. Therefore, it is also possible to use the start of this shooting interval as evaluation point of the constraint functions. This is reasonable because algebraic variables generally change discontinuously at the start of the next shooting interval and the z value reported at the start of the simulation provides a value that then continuously evolves over the shooting interval. Thus, an integrator for DAEs typically reports an approximation of z at t_n or t_{n+1}, although any other time point in $[t_n, t_{n+1}]$ is possible. In the `acados` framework, the convention is to always report the algebraic variables at the start of the simulation. Thus, in the following this will be the focus.

- **Obtain z as solution of a nonlinear system:** One possibility to obtain an approximation of z at a certain point in time (either t_n or t_{n+1}) is to solve the following nonlinear system in $\bar{k}, \bar{z}$:

$$f^{\text{impl}}(\bar{k}, \bar{x}, \bar{z}, u), \tag{2.19}$$

  whereby $\bar{x}$ is either x_n or x_{n+1} and $\bar{z}$ is the reported value. The Newton-scheme for this nonlinear system can be initialized efficiently using $\bar{k}^0 = k_1$ and $\bar{z}^0 = z_1$ or k_s, z_s respectively if z should be reported at the start or end of the simulation. Note that if the Butcher tableau contains the time point at which z is to be reported, we do not have to solve the extra nonlinear system (2.19), because $\bar{z}$ is part of the solution of (2.18a).

- **Obtain z by an interpolation formula:** After solving (2.18a), the values z_i for $i = 1, \ldots, s$ are available. In MPC, we most often have a constant control on each shooting interval. Therefore, the algebraic variables are given by a continuous function and it makes sense to approximate it by the interpolating polynomial corresponding to the nodes $(t_{n,i}, z_i)$ for $i = 1, \ldots, s$. This polynomial can be evaluated efficiently at the desired time point by the Neville-scheme, see e.g. [22].

- **Using a consistent initial value of z:** Another approach of generalizing RK methods to DAE systems assumes a consistent initial value pair (x_n, z_n). In the context of a semi-explicit DAE (2.5a) this means $g(x_n, z_n, u) = 0$. Hereby, the algebraic variables z are modeled just as the differential states using $k_j^z \in \mathbb{R}^{n_z}$

for $j = 1, \ldots, s$. We will not use this approach, as the algebraic states often change discontinuously at the start of the simulation due to the discontinuous change of the controls. A detailed formulation can be found in [19, (4.6)].

2.2.7 Applying Multiple Steps

In the context of multiple shooting, the integration method applied to each shooting interval often consists of a multiple step Runge-Kutta method. Assuming the shooting interval is $[t_n, t_{n+1}]$, then the multiple step RK method can be written as

$$G_n(w_n, K_n) = \begin{bmatrix} g_n^1(w_n, K_n^1) \\ \vdots \\ g_n^{n_{\text{steps}}}(w_n, K_n^1, \ldots, K_n^{n_{\text{steps}}}) \end{bmatrix} = 0, \tag{2.20a}$$

$$\text{where} \quad g_n^j(\cdot) = \begin{bmatrix} f^{\text{impl}}(k_{n,1}^j, x_n^{j-1} + T_{\text{int}} \sum_{r=1}^s a_{1r} k_{n,r}^j, z_{n,1}^j, u_n) \\ \vdots \\ f^{\text{impl}}(k_{n,s}^j, x_n^{j-1} + T_{\text{int}} \sum_{r=1}^s a_{sr} k_{n,r}^j, z_{n,s}^j, u_n) \end{bmatrix}. \tag{2.20b}$$

Here, the integration variables $k_{n,r}^j \in \mathbb{R}^{n_x}$ and $z_{n,r}^j \in \mathbb{R}^{n_z}$ for the shooting interval n and RK step j are collected in $K_n = \left(K_n^1, \ldots, K_n^{n_{\text{steps}}}\right)$ with $K_n^j = (k_{n,1}^j, z_{n,1}^j, \ldots, k_{n,s}^j, z_{n,s}^j)$ for $n = 0, \ldots, N-1$ and $j = 1, \ldots, n_{\text{steps}}$. Additionally, the shorthand $w_n = (x_n, u_n)$ is used to gather the initial state and control input.
The intermediate state values x_n^j are defined as

$$x_n^j = x_n^{j-1} + T_{\text{int}} \sum_{r=1}^s b_r k_{n,r}^j, \quad j = 1, \ldots, n_{\text{steps}},$$

where $x_n^0 := x_n$.

2.2.8 Reusing Jacobian Matrices

Within IRK implementations, it is quite common to reuse the Newton matrix and its factorization over multiple Newton iterations [12, p.83]. It is even possible to reuse the same Newton matrix over multiple integration steps, but this should be done carefully, since outdated Jacobian information can result in bad convergence properties.
Using the notation from Section 2.2.7, the reuse of the Jacobian over multiple steps can be formulated as follows [38]:
Within the multiple step IRK method, use the Jacobian $\frac{\partial g_n^1}{\partial K_n^1}$ as a Newton matrix in all Newton iterations corresponding to K_n^j for all $j = 1, \ldots, n_{\text{steps}}$.
If an integrator with sensitivity propagation via IND is applied, see Section 2.3, it

is good practice to use the following approach. Within the first step, two Jacobian matrices are evaluated and factorized, one for the Newton iterations and one for the sensitivity propagation via IND. The latter one is reused within the Newton iterations of the next integration step. For the IND, we calculate and factorize a new Newton matrix. Using this approach only $n_{\text{steps}} + 1$ matrices have to be factorized and a good convergence rate is maintained, [39], [12, p.88f].
Although the integrators compared in Chapter 5 support the option to reuse the Jacobian matrices as described above, it was not used within the numerical experiments presented there. The reasoning for this decision was to focus on very accurate results to verify that both schemes converge to the same solution. However, the integration scheme presented in Chapter 3 should be tested further using this option.

2.3 Sensitivity propagation

In the MPC framework, numerical optimization methods (e.g. SQP) are applied to a NLP which arises from an OCP via discretization. Within the numerical optimization method, one has to solve IVPs, which `acados` realizes by calling an *integrator*. These IVPs typically correspond to one shooting interval. Thus, the control input is assumed to be constant and we will write u instead of u_n in this context. Integrators do not only have to report the numerical solution of the IVP (2.6) at the final time T, which we denote as $x_T(x_0, u)$. Additionally, the sensitivities of the simulation result with respect to the initial state value x_0 and the control inputs u

$$\frac{\partial x_T(x_0, u)}{\partial (x_0, u)} \in \mathbb{R}^{n_{\mathrm{x}} \times (n_{\mathrm{x}} + n_{\mathrm{u}})}, \tag{2.21}$$

called *forward sensitivities* should be provided.
In the case of adjoint based Newton-type optimization methods, the integrator should report the *adjoint first order sensitivities*

$$\frac{\partial x_T(x_0, u)}{\partial (x_0, u)}^{\top} \lambda \in \mathbb{R}^{n_{\mathrm{x}} + n_{\mathrm{u}}}, \tag{2.22}$$

instead of the forward sensitivities (2.21), where $\lambda \in \mathbb{R}^{n_{\mathrm{x}}}$ denotes the *adjoint* or *backward seed*.

Higher order sensitivities Numerical optimization methods can also make use of higher order sensitivities. The theory on higher order sensitivity propagation is outside the scope of this work. However, during the time of this thesis project, the standard IRK integrator was extended by the author such that it supports also second

order sensitivity propagation, using the symmetric Hessian propagation presented in [40]. A detailed discussion on higher order sensitivity propagation can be found in [19, Ch.3] or [34].

2.3.1 Sensitivities for Direct Optimal Control – An Overview

In the following sections, different approaches to generate the first order sensitivities required within a direct optimal control algorithm are presented. Therefore, we assume that the model functions in the IVP are sufficiently smooth. Sensitivity propagation techniques can be categorized on whether they are based on a *differentiate-then-discretize* or *discretize-then-differentiate* type of approach. A brief discussion of these approaches is given in the following.
The idea of the differentiate-then-discretize approach is to extend the ODE model with variational differential equations (VDE) which describe how the (adjoint) sensitivities in (2.21), respectively (2.22), evolve in time. This system of variational equations can then be simulated using any numerical integration method with arbitrary accuracy. The fact that the VDE system propagates the sensitivities of the exact solution seems desirable at first sight. However, in the context of embedded optimization the mismatch between exact and numerical solution is often not negligible and it is desirable to propagate the sensitivities of the numerical solution instead, such that the simulation result and the derivatives in the Newton-type optimization algorithm are consistent.
In the discretize-then-differentiate approach the differentiation task is performed after the discretization, resulting in a discrete-time sensitivity propagation. The remarks in the above paragraph motivate the usage of this approach in direct optimal control, which therefore will be the focus in the remainder of this thesis.

Finite Differences and External Numerical Differentiation One classic approach to generate sensitivities, like the ones in (2.21), is to use Finite Differences. Here, the integrator is considered as a black-box function and the *finite differences* (FD) technique is used to approximate a directional derivative of the simulation result $x_T(x_0, u)$. Using the shorthand $w = (x_0, u)$, the directional derivative $\frac{\partial x_T(w)}{\partial w}\bar{w} \in \mathbb{R}^{n_x}$, where $\bar{w}$ is the direction, can be approximated by

$$\frac{\partial x_T(w)}{\partial w}\bar{w} \approx \frac{x_T(w + \delta\bar{w}) - x_T(w)}{\delta}. \tag{2.23}$$

Note, that this derivative approximation is strongly dependent on the perturbation value δ. Even for an optimal choice of δ, which depends on the problem functions, the loss of accuracy is about half the valid digits of the underlying function evaluation [34].

As a rule of thumb, one can chose $\delta \approx \sqrt{\text{TOL}}$, where TOL is the machine precision, see [12]. Note that for an evaluation of the full Jacobian $\frac{\partial x_T(w)}{\partial w}$ the integrator has to be called $n_\text{x} + n_\text{u} + 1$ times. The usage of FD is typically referred to as *external numerical differentiation* (END), because the differentiation is performed outside the integrator using multiple calls. The most important benefit of END is that it is very easy to implement.

Internal Numerical Differentiation The alternative approach to END is *Internal Numerical Differentiation* (IND), where the sensitivity propagation is performed within the integrator. The main benefit of IND is that it can also handle integration schemes with adaptive components, such as step size, order, iteration matrices and the number of Newton-type iterations, which should not be combined with END, as these adaptive components lead to discontinuities in the map $(x_0, u) \mapsto x_T(x_0, u)$. The idea is to freeze the adaptive components to obtain a fixed integrator function, which then can be differentiated using either AD or FD.
IND techniques can be further distinguished by whether they use a direct or an iterative approach. Hereby, the latter should only be used for implicit integration schemes with inexact or fixed iteration matrices.

2.3.2 Direct IND

Let us consider a constant control input u on the simulation interval $[0, T]$ which is divided into n_steps equidistant subintervals $[t_n, t_{n+1}]$ with $t_n = nT_\text{int}$ and $T_\text{int} = \frac{T}{n_\text{steps}}$. Using the general formulation (2.8), one step of an integration method applied to $[t_n, t_{n+1}]$ with initial state x_n reads as

$$0 = G(x_n, K_n, u) \tag{2.24a}$$

$$x_{n+1} = \Psi(x_n, K_n, u), \tag{2.24b}$$

where K_n gathers the internal integration variables, implicitly given by $G(\cdot)$, such that the Jacobian $\frac{\partial G}{\partial K}(\cdot)$ needs to be invertible. For example, in case of an s-stage RK method applied to a DAE system, it reads as $K_n = (k_{n,1}, z_{n,1} \ldots, k_{n,s}, z_{n,s})$. This special case will be discussed in the last part of this section. However, the general derivation is needed to derive the efficient structure sensitivity generation of the GNSF-IRK scheme presented in Chapter 3.

Forward Propagation The idea is to obtain the first order derivatives $\frac{\partial x_{n+1}}{\partial x_0}, \frac{\partial x_{n+1}}{\partial u}$ based on the corresponding sensitivity results of the previous integration step using

the implicit function theorem.

$$\begin{aligned}\left[\frac{\partial x_{n+1}}{\partial x_0} \quad \frac{\partial x_{n+1}}{\partial u}\right] &= \left[\frac{\partial \Psi_n}{\partial x}\frac{\partial x_n}{\partial x_0} \quad \frac{\partial \Psi_n}{\partial x}\frac{\partial x_n}{\partial u} + \frac{\partial \Psi_n}{\partial u}\right] + \frac{\partial \Psi_n}{\partial K}\left[\frac{\partial K_n}{\partial x_0} \quad \frac{\partial K_n}{\partial u}\right] \\ \left[\frac{\partial K_n}{\partial x_0} \quad \frac{\partial K_n}{\partial u}\right] &= -\frac{\partial G_n}{\partial K}^{-1}\left[\frac{\partial G_n}{\partial x}\frac{\partial x_n}{\partial x_0} \quad \frac{\partial G_n}{\partial x}\frac{\partial x_n}{\partial u} + \frac{\partial G_n}{\partial u}\right],\end{aligned} \tag{2.25}$$

where the compact notation $\frac{\partial G_n}{\partial K} = \frac{\partial G}{\partial K}(x_n, K_n, u)$, $\frac{\partial G_n}{\partial x} = \frac{\partial G}{\partial x}(x_n, K_n, u)$ and $\frac{\partial G_n}{\partial u} = \frac{\partial G}{\partial u}(x_n, K_n, u)$ is used and similarly for the function $\Psi(\cdot)$. The required derivative evaluations, e.g. of $\frac{\partial G_n}{\partial x}$, $\frac{\partial G_n}{\partial u}$, are typically performed efficiently using AD techniques. Most often, the main computational effort of the direct approach is the factorization of the Jacobian $\frac{\partial G_n}{\partial K}$ and the solution of the corresponding linear system. The current forward sensitivities are stored in a matrix of the form $\left[\frac{\partial x_n}{\partial x_0} \quad \frac{\partial x_n}{\partial u}\right]$ which is initialized with $\left[\frac{\partial x_0}{\partial x_0} \quad \frac{\partial x_0}{\partial u}\right] = [\mathbb{1} \quad \mathbb{0}]$ and gets updated in each step.

Adjoint Propagation In this paragraph, we derive a general way to propagate adjoint sensitivities, which were introduced in (2.22). Let us define the adjoint variables

$$\lambda_n^w = \left(\frac{\partial w_{n_{\text{steps}}}}{\partial w_n}\right)^\top \bar{\lambda} = \begin{bmatrix} \frac{\partial x_{n_{\text{steps}}}^\top}{\partial x_n} & \mathbb{0} \\ \frac{\partial x_{n_{\text{steps}}}^\top}{\partial u} & \mathbb{1} \end{bmatrix} \bar{\lambda}, \quad \text{for all} \quad n = 0, \ldots, n_{\text{steps}}, \tag{2.26}$$

whereby $\bar{\lambda}$ is the adjoint seed which of dimension $n_{\text{x}} + n_{\text{u}}$ in contrast to (2.22), where it is of dimension n_{x}. This should clarify that for multiple step integrators, the adjoint seed for step n will be the result of the adjoint sensitivity propagation in step $n+1$, which is of dimension $n_{\text{x}} + n_{\text{u}}$. From this point of view, the first adjoint seed $\bar{\lambda} \in \mathbb{R}^{n_{\text{x}}+n_{\text{u}}}$ is just a special case in which the last n_{u} components of the seed are zero.

As in the previous paragraph, we regard the general integration scheme (2.24) and assume a constant control input u for the integration steps $n = 1, \ldots, n_{\text{steps}}$. This said, we can write

$$w_{n+1} = \begin{bmatrix} x_{n+1} \\ u \end{bmatrix} = \begin{bmatrix} \Psi(x_n, K_n, u) \\ u \end{bmatrix} =: \widetilde{\Psi}(x_n, K_n, u), \tag{2.27}$$

where the extended output function $\widetilde{\Psi}$ is defined to arrive at a compact notation. By applying $\frac{\partial}{\partial w_n}$ to (2.27) and the integration equations (2.24a), one obtains:

$$\frac{\partial w_{n+1}}{\partial w_n} = \frac{\partial \widetilde{\Psi}_n}{\partial w} + \frac{\partial \widetilde{\Psi}_n}{\partial K}\frac{\partial K_n}{\partial w_n}, \tag{2.28a}$$

$$0 = \frac{\partial G_n}{\partial w} + \frac{\partial G_n}{\partial K}\frac{\partial K_n}{\partial w_n}, \tag{2.28b}$$

where the partial derivatives are notated as in the previous paragraph.
Note that the partial derivatives of $\widetilde{\Psi}$ read as

$$\frac{\partial \widetilde{\Psi}_n}{\partial w} = \begin{bmatrix} \frac{\partial \Psi_n}{\partial x} & \frac{\partial \Psi_n}{\partial u} \\ \mathbb{0} & \mathbb{1} \end{bmatrix}, \quad \frac{\partial \widetilde{\Psi}_n}{\partial K} = \begin{bmatrix} \frac{\partial \Psi_n}{\partial K} \\ \mathbb{0} \end{bmatrix},$$

and that this structure should always be exploited. After transposing (2.28a) and multiplying with λ_{n+1}^w, we get

$$\frac{\partial w_{n+1}}{\partial w_n}^\top \lambda_{n+1}^w = \frac{\partial \widetilde{\Psi}_n}{\partial w}^\top \lambda_{n+1}^w + \frac{\partial K_n}{\partial w_n}^\top \frac{\partial \widetilde{\Psi}_n}{\partial K}^\top \lambda_{n+1}^w. \tag{2.29}$$

Now let us introduce λ_{n+1}^K as the solution of

$$0 = \frac{\partial \widetilde{\Psi}_n}{\partial K}^\top \lambda_{n+1}^w + \frac{\partial G_n}{\partial K}^\top \lambda_{n+1}^K \tag{2.30}$$

and note that (2.28b) implies $\frac{\partial K_n}{\partial w_n}^\top = -\frac{\partial G_n}{\partial w}^\top \frac{\partial G_n}{\partial K}^{-\top}$. Furthermore, the chain rule gives us $\frac{\partial w_{n+1}}{\partial w_n}^\top \lambda_{n+1}^w = \lambda_n^w$. Therefore we obtain

$$\lambda_n^w = \frac{\partial \widetilde{\Psi}_n}{\partial w}^\top \lambda_{n+1}^w + \frac{\partial G_n}{\partial w}^\top \lambda_{n+1}^K. \tag{2.31}$$

In the adjoint IND scheme for $n = n_{\text{steps}} - 1, \ldots, 0$ first equation (2.30) is solved for λ_{n+1}^K such that equation (2.31) can be evaluated. In the end, we obtain the desired sensitivity $\lambda_0^w = \frac{\partial w_{n_{\text{steps}}}}{\partial w_0}^\top \bar{\lambda}$.
Note that in a classic RK scheme (2.31) can be simplified further, because $\frac{\partial \Psi_n}{\partial x} = \mathbb{1}$.

2.3.3 Sensitivity Propagation for Standard RK methods

In this section the direct IND scheme is more concretely described for the case of applying an s-stage RK method to a fully implicit index-1-DAE. In this case, the

functions Ψ, G read as

$$0 = G(x_n, K_n, u) = \left[f^{\text{impl}}(k_i, x_n + T_{\text{int}} \sum_{j=1}^{s} a_{ij} k_j, z_i, u) \right]_{i=1,\dots,s} \tag{2.32}$$

$$x_{n+1} = \Psi(x_n, K_n, u) = x_n + T_{\text{int}} \sum_{j=1}^{s} b_j k_j. \tag{2.33}$$

The integration variables in this section are gathered as $K_n = (k_1, \dots, k_s, z_1, \dots, z_s)$. Observe that the function Ψ is linear, thus the corresponding derivatives are trivially given by

$$\frac{\partial \Psi}{\partial x} = \mathbb{1}, \quad \frac{\partial \Psi}{\partial u} = \mathbb{0}, \quad \frac{\partial \Psi}{\partial K} = \left[\left[\begin{array}{ccc} T_{\text{int}} b_j & & 0 \\ & \ddots & \\ 0 & & T_{\text{int}} b_j \end{array} \right]_{j=1,\dots,s} \middle| \, \mathbb{0} \right]. \tag{2.34}$$

The Newton matrix can be written as

$$\frac{\partial G}{\partial K} = \left[\begin{array}{ccc|c} T_{\text{int}} a_{11} \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial \dot{x}} & & T_{\text{int}} a_{1s} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial z} & & \mathbb{0} \\ & \ddots & & & \ddots & \\ T_{\text{int}} a_{s1} \frac{\partial f_s}{\partial x} & & T_{\text{int}} a_{ss} \frac{\partial f_s}{\partial x} + \frac{\partial f_s}{\partial \dot{x}} & \mathbb{0} & & \frac{\partial f_s}{\partial z} \end{array} \right], \tag{2.35}$$

where the shorthand $f_i = f^{\text{impl}}(k_i, x_n + T_{\text{int}} \sum_{j=1}^{s} a_{ij} k_j, z_i, u)$ is used for $i = 1, \dots, s$. The remaining derivatives of G are simply

$$\frac{\partial G}{\partial x} = \left[\frac{\partial f_i}{\partial x} \right]_{i=1,\dots,s}, \quad \frac{\partial G}{\partial u} = \left[\frac{\partial f_i}{\partial u} \right]_{i=1,\dots,s}. \tag{2.36}$$

Direct Forward Propagation The forward propagation technique derived in Section 2.3.2 can be simplified for this standard IRK method, and the resulting procedure is described in Algorithm 1.

Direct Adjoint Propagation Similarly, the adjoint propagation technique derived in Section 2.3.2 can be simplified for this standard IRK method, and the resulting procedure is described in Algorithm 2.

Algorithm 1: Direct Forward Sensitivity Propagation for IRK

Input: Implicit index-1 DAE function $f^{\text{impl}}(\dot{x}, x, z, u)$, current forward sensitivities $S_{\text{forw}} \approx \begin{bmatrix} \frac{\partial x_n}{\partial x_0} & \frac{\partial x_n}{\partial u} \end{bmatrix}$

`Preparation and Evaluation`

Evaluate $\frac{\partial f_i}{\partial \dot{x}, x, z, u}$, for $i = 1, \ldots, s$ ▷ Using AD

Set up $\frac{\partial G}{\partial w}, \frac{\partial G}{\partial K}$ ▷ (2.35), (2.36)

$\frac{\partial G}{\partial w_0} \leftarrow \left[\frac{\partial G}{\partial x} S_{\text{forw}}\right] + \begin{bmatrix} \mathbb{0} & \frac{\partial G}{\partial u} \end{bmatrix}$

`Solve linear system`

$\frac{\partial G}{\partial K} \frac{\partial K}{\partial w_0} = \frac{\partial G}{\partial w_0}$

`Update Sensitivities`

$S_{\text{forw}} \leftarrow S_{\text{forw}} + T_{\text{int}} \sum_{i=1}^{s} b_i \frac{\partial k_i}{\partial w_0}$ ▷ New Sensitivities

Algorithm 2: Direct Adjoint Sensitivity Propagation for IRK

Input: Implicit index-1 DAE function $f^{\text{impl}}(\dot{x}, x, z, u)$, current adjoint sensitivities λ_{n+1}^{w}

`Preparation and Evaluation`

Evaluate $\frac{\partial f_i}{\partial \dot{x}, x, z, u}$, for $i = 1, \ldots, s$ ▷ Using AD

Note: K_n have to be stored in simulation

Set up $\frac{\partial G}{\partial w}, \frac{\partial G}{\partial K}$ ▷ (2.35), (2.36)

`Solve linear system`

$$\lambda_{n+1}^{K} \leftarrow \left(\frac{\partial G_n}{\partial K}\right)^{-\top} \begin{bmatrix} T_{\text{int}} b_1 \lambda_{n+1}^{x} \\ \vdots \\ T_{\text{int}} b_s \lambda_{n+1}^{x} \\ \mathbb{0} \end{bmatrix}$$

`Update Sensitivities`

Update $\lambda_n^w = \lambda_{n+1}^w + \frac{\partial G}{\partial w} \lambda_{n+1}^K$ ▷ New Sensitivities

2.3.4 Sensitivity Propagation for Algebraic Variables

As mentioned in Section 2.2.6, algebraic variables are likely to be part of constraint or objective functions. In this case, the MPC algorithm will not only need a value for z but the integrator should additionally provide the sensitivities of this value with respect to the initial state and the control, i.e.

$$\frac{\partial z(x_0, u)}{\partial (x_0, u)} \in \mathbb{R}^{n_{\text{z}} \times (n_{\text{x}} + n_{\text{u}})}. \tag{2.37}$$

Let us assume a fully implicit DAE (2.3), which is simulated by an IRK scheme that reports z_0, an approximation of z at the start of the simulation interval ap-

plying an interpolation formula. In this case, the integrator should additionally provide an approximation of $\frac{\partial z_0(x_0,u)}{\partial(x_0,u)}$. Assuming a consistent tuple $(\dot{x}_0, x_0, z_0, u)$, i.e. $f^{\text{impl}}(\dot{x}_0, x_0, z_0, u) = 0$, we can apply the implicit function theorem and obtain

$$0 = \frac{\partial f^{\text{impl}}}{\partial(x,u)}(\dot{x}_0, x_0, z_0, u) + \frac{\partial f^{\text{impl}}}{\partial(\dot{x},z)}(\dot{x}_0, x_0, z_0, u)\frac{\partial(\dot{x}_0, z_0)}{\partial(x_0,u)}. \tag{2.38}$$

Therefore, the sensitivities of the algebraic variables can be obtained by solving

$$\frac{\partial(\dot{x}_0, z_0)}{\partial(x_0,u)} = -\left(\frac{\partial f^{\text{impl}}}{\partial(\dot{x},z)}(\dot{x}_0, x_0, z_0, u)\right)^{-1}\frac{\partial f^{\text{impl}}}{\partial(x,u)}(\dot{x}_0, x_0, z_0, u) \tag{2.39}$$

and taking the corresponding submatrix of the solution.
In practice, we need to provide a value for $\dot{x}$ to evaluate f^{impl} at time t_0. This can be done just like discussed for z in Section 2.2.6, either by applying an interpolation formula on the points $(t_i, k_i), i = 1, \ldots, s$, or if a more exact approximation is desired, one can apply Newton iterations on $\dot{x}_0, z_0$ and the function $f^{\text{impl}}(\cdot, x_0, \cdot, u) = 0$. The computational cost of this sensitivity propagation is either dominated by the evaluation of the derivatives of f^{impl} or the solution of the linear system.

3 Structure Exploiting IRK Schemes

In this chapter, we want to introduce and discuss the concept of structure exploiting integrators, more specifically IRK schemes.

Motivation As the integration part is a key element in any NMPC scheme, that often dominates the computational cost in a Newton-type optimization algorithm, tuning the integrator is essential. Typically, the main computational cost within an IRK scheme is the factorization of the Newton matrix and the solution of the corresponding linear systems. In order to reduce computation time, different approaches have been proposed on how to treat this nonlinear system. This chapter aims to discuss the most important ideas and presents a novel scheme (GNSF-IRK).
The chapter is organized as follows. In Section 3.1, a short overview on previously used dynamic system structures is given. In Section 3.2, we propose a generalization of these dynamic system structures which we call "Generalized Nonlinear Static Feedback structure", short GNSF, see Equation (3.12). In Section 3.3, an efficient IRK scheme, so-called GNSF-IRK, is derived to simulate GNSF structured models and propagate first order sensitivities efficiently. Finally, in Section 3.4, the GNSF-IRK Algorithm is compared with earlier approaches and possible extensions are presented.

3.1 State of the Art and Previous Work

This section aims to give a brief overview on the previous work on approaches to exploit the structure of the Newton matrix within an IRK scheme.
Generally, there are two ways to exploit the structure of the linear system that is typically solved within an IRK scheme.
First, there is the classical idea to use a well-suited approximation of the Newton matrix, for which the derivatives of the DAE are only evaluated once. The main idea is that the IRK equations of each stage have a similar structure and the corresponding Newton matrix can be factorized relatively cheaply. This concept is presented in Section 3.1.1.
The second approach is to exploit the structure present within the dynamic system and was suggested more recently. The ideas presented in Section 3.1.2 and Section 3.1.3 sum up the previous work on this concept. The scheme that is derived later in this chapter follows this approach.
The main idea of our approach is that given a dynamic system structure with an "isolated" nonlinear subsystem, one can isolate the corresponding integration equations and solve them separately. Using this technique, an equivalent integration scheme

can be derived which performs Newton iterations on a lower dimensional space and makes them computationally much cheaper. One understands this idea most easily regarding the three-stage dynamic system structure in Section 3.1.2, as the "isolated" nonlinear subsystem is most obvious there.
When modeling a physical system, parts of the dynamics are often linear and there are just some nonlinear terms, which cause the need for an implicit integrator.

3.1.1 Structure Exploitation within an Inexact Newton Scheme

The idea has been initially described by Butcher and Bickart, in [41] and [42] respectively, and further discussed and extended by many others, e.g. [43; 44; 45]. This section mainly follows the notation in [38], because it uses a general DAE framework and focuses on the MPC context.

Simplified Newton For the sake of simplicity, let us regard an IRK scheme with $n_{\text{steps}} = 1$ and consider the special case $s = 3$, for which the essential concepts can be illustrated nicely.
The exact Jacobian of $G_n(w_n, K_n)$ in (2.20) reads as

$$\frac{\partial G_n}{\partial K} = \begin{bmatrix} H_1 + T_{\text{int}} a_{11} J_1 & T_{\text{int}} a_{12} J_1 & T_{\text{int}} a_{13} J_1 \\ T_{\text{int}} a_{21} J_2 & H_2 + T_{\text{int}} a_{22} J_2 & T_{\text{int}} a_{23} J_2 \\ T_{\text{int}} a_{31} J_3 & T_{\text{int}} a_{32} J_3 & H_3 + T_{\text{int}} a_{33} J_3 \end{bmatrix}, \tag{3.1}$$

where $H_j = \begin{bmatrix} \frac{\mathrm{d} f_{n,j}}{\mathrm{d}\dot{x}} & \frac{\mathrm{d} f_{n,j}}{\mathrm{d}z} \end{bmatrix}$ and $J_j = \begin{bmatrix} \frac{\mathrm{d} f_{n,j}}{\mathrm{d}x} & \mathbb{0} \end{bmatrix}$ are defined using the shorthand $f_{n,j} = f^{\text{impl}}(k_{n,j}, x_n + T_{\text{int}} \sum_{r=1}^{s} a_{jr} k_{n,r}, z_{n,j}, u_n)$ for $j = 1, \ldots, s$.
The idea of the *Simplified Newton* iteration for IRK methods is to use a specific approximation M_i of the Jacobian in (3.1), where the partial derivatives of f^{impl} are only evaluated once. The Jacobian approximation is constructed as

$$M_n = \mathbb{1}_3 \otimes H + T_{\text{int}} A_{\text{but}} \otimes J = \begin{bmatrix} H + T_{\text{int}} a_{11} J & T_{\text{int}} a_{12} J & T_{\text{int}} a_{13} J \\ T_{\text{int}} a_{21} J & H + T_{\text{int}} a_{22} J & T_{\text{int}} a_{23} J \\ T_{\text{int}} a_{31} J & T_{\text{int}} a_{32} J & H + T_{\text{int}} a_{33} J \end{bmatrix}, \tag{3.2}$$

where $\otimes$ denotes the Kronecker product and the matrices H, J are some evaluation of $\begin{bmatrix} \frac{\mathrm{d} f^{\text{impl}}}{\mathrm{d}\dot{x}} & \frac{\mathrm{d} f^{\text{impl}}}{\mathrm{d}z} \end{bmatrix}$ and $\begin{bmatrix} \frac{\mathrm{d} f^{\text{impl}}}{\mathrm{d}x} & \mathbb{0} \end{bmatrix}$ respectively. There are different recommendations for the evaluation point of H, J. In general, one can choose any intermediate or recent point [45]. However, the previous point seems reasonable in case of an explicit ODE [44] and the first stage point in case of an implicit ODE or DAE [38].

In a Newton-type iteration on (2.20), one has to solve

$$(\mathbb{1}_3 \otimes H + T_{\text{int}} A_{\text{but}} \otimes J)\Delta K_n = -G_n, \tag{3.3}$$

for ΔK_n to update $K_n \leftarrow K_n + \Delta K_n$.
Now, we have to assume that the Butcher matrix $A_{\text{but}} \in \mathbb{R}^{3\times 3}$ is invertible and that there is decomposition $A^{-1} = V\Lambda V^{-1}$ with a block-diagonal matrix Λ.
It is typical for A^{-1} to have one real eigenvalue γ and a complex conjugate eigenvalue pair $\alpha \pm i\beta$, for example for Radau IIA [31]. Using this decomposition, we can premultiply (3.3) by $(T_{\text{int}} A_{\text{but}})^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}}$ and obtain

$$\left(\tilde{\Lambda} \otimes H + \mathbb{1}_3 \otimes J\right) \Delta\tilde{K}_i = -\left(\tilde{\Lambda} V^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}}\right) G_i, \tag{3.4}$$

where $\Delta\tilde{K}_i = \left(V^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}}\right) \Delta K_i$ and $\tilde{\Lambda} = \frac{\Lambda}{T_{\text{int}}}$.
Note that the matrix which has to be factorized when solving (3.4) has the following structure:

$$\tilde{\Lambda} \otimes H + \mathbb{1}_3 \otimes J = \begin{bmatrix} \tilde{\gamma}H + J & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \tilde{\alpha}H + J & -\tilde{\beta}H \\ \mathbb{0} & \tilde{\beta}H & \tilde{\alpha}H + J \end{bmatrix}, \tag{3.5}$$

where the scaled eigenvalues are defined as $\tilde{\gamma} = \frac{\gamma}{T_{\text{int}}}$, $\tilde{\alpha} = \frac{\alpha}{T_{\text{int}}}$ and $\tilde{\beta} = \frac{\beta}{T_{\text{int}}}$.
Thus, we can split the linear system (3.4) into two subsystems of dimension $n_{\text{x}} + n_{\text{z}}$ and $2(n_{\text{x}} + n_{\text{z}})$. The latter one can be transformed into a $(n_{\text{x}} + n_{\text{z}})$-dimensional complex system, as described in [31, p.122]. These transformations can also be deployed for both the linear system of the Newton iteration and the one to obtain the sensitivities, i.e. (2.25).

Single Newton The Single Newton method uses the same approximation of the Newton matrix (3.5). However, in this scheme the matrix $(A_{\text{but}})^{-1}$ must have only one real eigenvalue γ. In this case, the Newton matrix (3.5) is even more structured, namely

$$\begin{bmatrix} \tilde{\gamma}H + J & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \tilde{\gamma}H + J & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \tilde{\gamma}H + J \end{bmatrix}. \tag{3.6}$$

Thus, the linear system (3.4) is equivalent to three separate systems with the same real coefficient matrix $\tilde{\gamma}H + J$.
However, for most high order IRK methods, $(A_{\text{but}})^{-1}$ does not have the desired

property [31]. Therefore, we typically have to use an approximation $\tilde{A}$ of A_{but}, which is selected such that its inverse has only one real eigenvalue γ. The matrix $\tilde{A}$ is regular and has a decomposition of the form

$$\tilde{A}^{-1} = \gamma W(\mathbb{1}_3 - E)W^{-1},$$

where E is a strictly lower triangular matrix. A detailed description on the construction of such a approximation of A_{but} can be found in [43].
Replacing A_{but} with such a matrix $\tilde{A}$ in (3.3) and premultiplying it with $(T_{\text{int}}\tilde{A})^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}}$, we obtain the following expression

$$(\mathbb{1}_3 \otimes (\tilde{\gamma}H + J))\Delta\hat{K}_i = -(\tilde{\gamma}(\mathbb{1}_3 - E)W^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}})G_i + (E \otimes \tilde{\gamma}H)\Delta\hat{K}_i, \quad (3.7)$$

where the shorthands $\tilde{\gamma} = \frac{\gamma}{T_{\text{int}}}$ and $\Delta\hat{K}_i = (W^{-1} \otimes \mathbb{1}_{n_{\text{x}}+n_{\text{z}}})\Delta K_i$ are used. In order to solve the linear system in (3.7) efficiently, we factorize the matrix $\tilde{\gamma}H + J$ and subsequently solve three subsystems of dimension $n_{\text{x}}+n_{\text{z}}$, which are separable because E is strictly lower diagonal.

3.1.2 A Three-Stage Dynamic Structure

To get an idea of how a specific dynamic system structure can be exploited within an integrator, let us first regard a differential equation with a three-stage dynamic system structure as proposed in [46]. In this type of model, the state vector is partitioned as $x = \left[x^{\text{LI}\top}, x^{\text{NL}\top}, x^{\text{LO}\top}\right]^\top \in \mathbb{R}^{n_{\text{x}}}$, with $x^{\text{LI}} \in \mathbb{R}^{n^{\text{LI}}}$, $x^{\text{NL}} \in \mathbb{R}^{n^{\text{NL}}}$, $x^{\text{LO}} \in \mathbb{R}^{n^{\text{LO}}}$ and the dynamics can be written as

$$\dot{x}^{\text{LI}} = A_{\text{LI}}x^{\text{LI}} + B_{\text{LI}}u \quad (3.8a)$$
$$\dot{x}^{\text{NL}} = f_{\text{NL}}(x^{\text{LI}}, x^{\text{NL}}, u) \quad (3.8b)$$
$$\dot{x}^{\text{LO}} = A_{\text{LO}}x^{\text{LO}} + f_{\text{LO}}(x^{\text{LI}}, x^{\text{NL}}, u). \quad (3.8c)$$

This system consists of the following three subsystems: the linear input system (3.8a), the nonlinear system (3.8b) and the linear output system (LOS) (3.8c). The "isolated" nonlinear subsystem mentioned initially in Section 3.1 is obviously (3.8b). The associated structure exploiting IRK scheme first solves the integration equations corresponding to the linear input system (3.8a) exactly in one Newton iteration. This is possible as they are linear and only depend on x^{LI} and the corresponding integration variables K^{LI}. As now K^{LI} is given, the algorithm can perform some Newton-type iterations (with dimension sn^{NL}) on the integration equations corresponding to (3.8b) and solve them for K^{NL} up to some precision. Finally, as now K^{LI} and K^{NL} are

available, the integration equations corresponding to (3.8c) can be solved easily. The nonlinear function f_{LO} only has to be evaluated at the stage points and subsequently the integration variables K^{LO} are defined by a linear system whose matrix can be precomputed and factorized off-line. Note that the structure can be exploited similarly in the sensitivity propagation.
The introduction above is meant to give a brief overview on how dynamic system structures can be exploited within IRK schemes. A more detailed explanation on this specific scheme can be found in [12, Ch.4] or [46].

3.1.3 Nonlinear Static Feedback Structure

In [47], a structure exploiting collocation scheme for a dynamic system, in the following called *Nonlinear Static Feedback* (NSF), structure has been presented

$$E\dot{x} = Ax + Bu + C\varphi\left(Dx + F\dot{x}, u\right), \tag{3.9}$$

where the differential states $x \in \mathbb{R}^{n_{\mathrm{x}}}$ and control inputs $u \in \mathbb{R}^{n_{\mathrm{u}}}$ are used. The NSF structured system is defined by the nonlinear function $\varphi : \mathbb{R}^{n_{\mathrm{in}}} \to \mathbb{R}^{n_{\mathrm{out}}}$ and the matrices $A, E \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{x}}}, B \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{u}}}$, $D, F \in \mathbb{R}^{n_{\mathrm{in}} \times n_{\mathrm{x}}}$ and $C \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{out}}}$, which are assumed to be constant. The scheme presented in [47] performs Newton iterations on a space of dimension $s\, n_{\mathrm{out}}$.

3.1.4 Comparing Structure Exploiting IRK Schemes

In this section, we want to formally define a relation between dynamic system structures for IRK schemes. Using this definition, the formulations in Section 3.1.2 and Section 3.1.3 are formally compared.

Definition 3.1 (Generalization relation between dynamic system structures for IRK schemes) Given two structured dynamic systems (S1), (S2) and corresponding structure exploiting IRK schemes, we say that (S1) is a *generalization* of (S2) if the following holds:
Any dynamic model $(M2)$ given in the form (S2) can be reformulated into a model $(M1)$ of the form (S1) such that

$$\dim(\mathrm{S1}, M1) \leq \dim(\mathrm{S2}, M2),$$

where $\dim(\mathrm{S}, M)$ denotes the dimension of the Newton matrix in the scheme associated with the dynamic structure (S) applied to model (M).

Remark 3.2 (Partial Order) The set of dynamic system structures associated with IRK schemes can be partially ordered by the relation defined in Definition 3.1. The relation could be denoted as $\succeq$. Reflexivity and transitivity are easy to show and antisymmetry holds defining equivalence through the relation itself.

Theorem 3.3 *The NSF structure* (3.9) *is a generalization of the structure given by the linear input* (3.8a) *and nonlinear subsystem* (3.8b) *combined.*

Proof. Given a system in the form (3.8) with empty x^{LO}, we can transcribe the model into the form (3.9) by setting

$$x = \begin{bmatrix} x^{\mathrm{LI}} \\ x^{\mathrm{NL}} \end{bmatrix}, \quad A = \begin{bmatrix} A_{\mathrm{LI}} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} \end{bmatrix}, \quad B = \begin{bmatrix} B_{\mathrm{LI}} \\ \mathbb{0} \end{bmatrix}, \quad C = \begin{bmatrix} \mathbb{0} \\ \mathbb{1} \end{bmatrix},$$
$$E = \mathbb{1}, \quad \varphi(x,u) = f_{\mathrm{NL}}(x^{\mathrm{LI}}, x^{\mathrm{NL}}, u).$$

Then the matrix to be factorized within both schemes will be of dimension sn^{NL}. □

Theorem 3.4 *The NSF structure* (3.9) *is NOT a generalization of the three-staged dynamic system structure* (3.8).

Proof. This statement can be verified by regarding the following simple dynamic system for $x \in \mathbb{R}^2$

$$\dot{x}_1 = x_1 \tag{3.10a}$$
$$\dot{x}_2 = \sin(x_1). \tag{3.10b}$$

This can be simulated by the IRK scheme corresponding to (3.8) with an empty nonlinear subsystem (3.8b), because (3.10a) is in the form of (3.8a) and (3.10b) is in the form of (3.8b). Thus, using this scheme, there is no matrix to be factorized online.
On the other hand, using this system with the IRK scheme corresponding to the NSF structure (3.9), the sine function has to be modeled by φ. Thus, the Newton matrix that has to be factorized is at least of dimension s. □

3.1.5 Nonlinear Static Feedback with Linear Output

We have seen that the NSF is a generalization of a linear input system coupled to a nonlinear system, but the corresponding IRK scheme cannot handle the linear output system efficiently. For this reason, at the beginning of the work with structure exploiting integrators, we regarded the following dynamic system structure. It consists

of a Nonlinear Static Feedback coupled to Linear Output subsystem and is thus referred to as NSFLO. It reads as

$$\begin{aligned} E\dot{x}^{[1]} &= A_1 x^{[1]} + Bu + C\phi\left(Dx^{[1]} + F\dot{x}^{[1]}, u\right), \\ \dot{x}^{[2]} &= A_2 x^{[2]} + f(x^{[1]}, u), \end{aligned} \tag{3.11}$$

where the differential states are denoted as $x = \left[{x^{[1]}}^\top, {x^{[2]}}^\top\right]^\top \in \mathbb{R}^{n_\mathrm{x}}$ with $x^{[1]} \in \mathbb{R}^{n_{\mathrm{x}1}}, x^{[2]} \in \mathbb{R}^{n_{\mathrm{x}2}}$ and the control inputs as $u \in \mathbb{R}^{n_\mathrm{u}}$. Additionally, the functions $\phi : \mathbb{R}^{n_\mathrm{in}} \to \mathbb{R}^{n_\mathrm{out}}, f : \mathbb{R}^{n_{\mathrm{x}1}+n_\mathrm{u}} \to \mathbb{R}^{n_{\mathrm{x}2}}$ and the matrices $A_1, E \in \mathbb{R}^{n_{\mathrm{x}1}\times n_{\mathrm{x}1}}, B \in \mathbb{R}^{n_{\mathrm{x}1}\times n_\mathrm{u}}$, $D, F \in \mathbb{R}^{n_\mathrm{in}\times n_{\mathrm{x}1}}, C \in \mathbb{R}^{n_{\mathrm{x}1}\times n_\mathrm{out}}$ and $A_2 \in \mathbb{R}^{n_{\mathrm{x}2}\times n_{\mathrm{x}2}}$ are used.
We were able to derive a structure exploiting IRK scheme for NSFLO, which first solves the integration equations for the NSF part and subsequently the ones corresponding to the LOS. The matrix to be factorized within a Newton-type iteration is of dimension $s\,n_\mathrm{out}$. This scheme is a special case of the one presented in Section 3.3 and is thus not presented.

Theorem 3.5 *The dynamic system structure* (3.11) *is a generalization of both* (3.9) *and* (3.8) *in the sense of Definition 3.1.*

Proof. The fact that it is a generalization of (3.9) is trivial using (3.11) with empty $x^{[2]}$.
For the generalization of (3.8), we refer to the previous proof and transcribe the LOS naturally. □

3.2 Generalized Nonlinear Static Feedback (GNSF)

The main goal of this work on structure exploiting integrators was to generalize the previously used structured dynamic systems in the sense of Definition 3.1. This generalization should also be able to handle index-1 DAEs. Finally, an efficient IRK scheme should be derived that exploits this structure. Within the implementation of the scheme, the number of function evaluations and the complexity of BLAS level 3 operations, such as matrix-matrix multiplications and linear system solves given a matrix factorization, should be kept at a minimum.
In the pursuit of this goal, we arrived at the Generalized Nonlinear Static Feedback Structure (GNSF), which is presented in this section.

3.2.1 GNSF Structured Dynamic System Model

Starting with a prototype of an IRK scheme using the (3.11) structure, we successively added more dependencies and modifications to the structured dynamic system. A detailed reasoning on the choice of the model structure is presented in the Subsection 3.2.3.
The final structured dynamic system model that will be discussed in the rest of this chapter is called *Generalized Nonlinear Static Feedback structure* (GNSF) and reads as follows:

$$E \begin{bmatrix} \dot{x}^{[1]} \\ z^{[1]} \end{bmatrix} = Ax^{[1]} + Bu + C\phi(L_{\dot{\mathrm{x}}}\dot{x}^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z^{[1]}, L_{\mathrm{u}}u) + c, \tag{3.12a}$$

$$E^{\mathrm{LO}} \begin{bmatrix} \dot{x}^{[2]} \\ z^{[2]} \end{bmatrix} = A^{\mathrm{LO}}x^{[2]} + f_{\mathrm{LO}}(\dot{x}^{[1]}, x^{[1]}, z^{[1]}, u). \tag{3.12b}$$

Here, we denote the 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 $u \in \mathbb{R}^{n_{\mathrm{u}}}$, the (nonlinear) functions $\phi : \mathbb{R}^{n_{\mathrm{y}}+n_{\hat{\mathrm{u}}}} \to \mathbb{R}^{n_{\mathrm{out}}}$ and $f_{\mathrm{LO}} : \mathbb{R}^{2n_{\mathrm{x}_1}+n_{\mathrm{z}_1}+n_{\mathrm{u}}} \to \mathbb{R}^{n_{\mathrm{x}_2}}$, which will be referred to as *nonlinearity* and *linear output function* 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*), 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 assumed to be constant and can be divided naturally as

$$E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix},\ A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix},\ B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix},\ C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix},\ c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \tag{3.13}$$

Additionally, 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 linear output type system (3.12b).
Deploying a standard IRK scheme on (3.12), a square matrix of dimension $s(n_{\mathrm{x}}+n_{\mathrm{z}})$ has to be factorized to perform Newton iterations on the nonlinear system (2.18a). In Section 3.3, an equivalent structure exploiting IRK scheme is presented, in which the matrix to be factorized is of size sn_{out}.

3.2.2 Requirements on the GNSF 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. In order to apply the IRK scheme derived in the following sections, we have to additionally assume that E_{11} and E_{22} are invertible matrices.

3.2.3 Motivation - Choice of the Model Structure

In this section, we want to motivate the choice of the GNSF dynamic system structure (3.12) and its usage with `acados`.

Generalization of former dynamic system structures As mentioned before, our goal was to use a structured dynamic system that is a generalization of the previously used structures in the sense of Definition 3.1. For the sake of completeness, we state this formally.

Theorem 3.6 *The dynamic system structure* (3.12) *is a generalization of* (3.11)*,* (3.9) *and* (3.8) *in the sense of Definition 3.1.*

Proof. One can reformulate a model in the form (3.11) into (3.12) with empty z and the matrices transcribed naturally as $E^{\mathrm{LO}} = \mathbb{1}$, $L_{\dot{\mathrm{x}}} = F, L_{\mathrm{x}} = D, L_{\mathrm{u}} = \mathbb{1}$, $c = \mathbb{0}$, the first generalization is obvious.
Referring to Theorem 3.5, the remaining follows as the generalization relation $\succeq$ is transitive, Remark 3.2. □

Treatment of implicit DAEs In the context of embedded optimization, the system of interest is often modeled by a fully implicit DAE. Therefore, structure exploiting integrators for `acados` should be able to handle these types of equations. If an algebraic variable z_i is given by an implicit function, i.e.

$$f_{z_i}(x, \dot{x}, z, u) = 0,$$

it can typically be included into (3.12) by adding a component to ϕ which contains $f_{z_i} - z_i$ and adding a 1 on the corresponding diagonal entry of E. This trick enables

us to transform algebraic equations into the NSF type system with invertible matrices E_{11} and E_{22}, which is formally stated in the following theorem.

Theorem 3.7 *Any index-1 DAE of the form* $0 = f^{\text{impl}}(\dot{x}, x, z, u)$ *can be reformulated in the GNSF format* (3.12) *such that the proposed structure exploiting IRK scheme can be used, i.e. the requirements stated in Section 3.2.2 hold.*

Proof. We reformulate the DAE only using the NSF type system (3.12a) such that $x^{[1]} = x,\ z^{[1]} = z$ and an empty LOS (3.12b), i.e. $n_{\mathrm{x}_2} = 0,\ n_{\mathrm{z}_2} = 0$. We define the model matrices as $E = \mathbb{1},\ A = \mathbb{0},\ B = \mathbb{0},\ C = \mathbb{1},\ c = \mathbb{0}$ and the nonlinearity function ϕ as

$$\phi(y, \hat{u}) = \alpha f^{\text{impl}}(\dot{x}, x, z, u) + \begin{bmatrix} \dot{x} \\ z \end{bmatrix},$$

for some $\alpha \in \mathbb{R}$, that is chosen later. The inputs of ϕ are just $y = \left[\dot{x}^\top, x^\top, z^\top\right]^\top$ and $\hat{u} = u$. This GNSF structured system is equivalent to the implicit DAE and fulfills the requirement that E_{11}, E_{22} and E^{LO} must be invertible. Additionally, the matrix

$$E - C\frac{\partial \phi}{\partial y}\begin{bmatrix} L_{\dot{\mathrm{x}}} & L_{\mathrm{z}} \end{bmatrix} = \mathbb{1} - \alpha \frac{\partial f^{\text{impl}}}{\partial [\dot{x}, z]}$$

must be invertible, which we can achieve by a good choice of α, i.e. such that the eigenvalues of $\alpha \frac{\partial f^{\text{impl}}}{\partial [\dot{x}, z]}$ are not one, referring to Lemma 3.9. □

Remark 3.8 For practical applications of the model reformulation in the previous proof, one should try to choose α such that the condition number of $\mathbb{1} - \alpha \frac{\partial f^{\text{impl}}}{\partial [\dot{x}, z]}$ is small and the matrix is well conditioned.

Lemma 3.9 Given a matrix $M \in \mathbb{R}^{n \times n}$ with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$, the matrix $\mathbb{1} + \alpha M$ with $\alpha \in \mathbb{R}$ has eigenvalues $1 + \alpha\lambda_i$ for $i = 1, \ldots, n$.

Proof. For every eigenvalue λ_i of M with corresponding eigenvector v_i, the following holds

$$(\mathbb{1} + \alpha M)v_i = \mathbb{1}v_i + \alpha M v_i = (1 + \alpha\lambda_i)v_i$$ □

Maximization of Dependencies - Preserving Structure Exploitation The main concept of GNSF-IRK is to simulate the subsystems (3.12a) and (3.12b) subsequently in each integration step and to exploit the linear dependencies in the

precomputation phase. We included as many affine dependencies in (3.12a) as possible, preserving the independence of (3.12b). Similarly, the function ϕ ought to be as general as possible, preserving the independence of $x^{[2]}, z^{[2]}$. Therefore, the nonlinearity function ϕ and the linear output function f_{LO} can depend on all variables $\dot{x}^{[1]}, x^{[1]}, z^{[1]}, u$.

Splitted Input in Nonlinearity Function Another feature is that we chose to split the input of the nonlinearity function ϕ into y and $\hat{u}$. The reason for this is that the derivatives of ϕ with respect to the controls are not needed within the Newton iterations (see (3.28) and (3.29)) but only for the sensitivity propagation. Splitting the input enables us to compute these derivatives only when needed.

Linear Input Matrices Note that in the formulation (3.12), we defined the nonlinearity ϕ as a function of $y := L_{\dot{\mathrm{x}}}\dot{x}^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z^{[1]}$ and $\hat{u} := L_{\mathrm{u}}u$, instead of using the full input $\dot{x}^{[1]}, x^{[1]}, z$ and u. This was inspired by the original NSF structure (3.9) from [47], which uses a similar formulation. However, this point was discussed controversially during the development of the GNSF-IRK scheme for `acados`. We concluded that the alternative approach of omitting these matrices could be beneficial within software frameworks different from `acados`.
A presentation of this approach as well as a comparison to our GNSF-IRK scheme is given in Section 3.4.4. To fully understand the discussion in Section 3.4.4, a more detailed knowledge of the derivation of GNSF-IRK is required, which is thus presented first.

3.3 The GNSF Structure Exploiting IRK Scheme

In this section an efficient IRK scheme exploiting the GNSF structured dynamic system (3.12) is derived. The scheme is referred to as GNSF-IRK throughout this thesis.

3.3.1 IRK Scheme - Idea

We use the IRK formulation in Equation (2.18) and apply a lifting-condensing technique similar to the one presented in [47] to solve the IRK equations corresponding to the NSF type part (3.12a). Subsequently, the IRK equations corresponding to the LOS (3.12b) are solved by exploiting their structure similarly to the approach in [46].

3.3.2 Structured IRK Equations

The nonlinear system of IRK equations (2.18a) corresponding to the GNSF structured dynamic system (3.12) reads as follows:

$$\begin{aligned} 0 &= E_{11}k_i^{[1]} + E_{12}z_i^{[1]} - A_1(x_n^{[1]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[1]}) - B_1 u \\ &- C_1\phi\left(L_{\dot{\text{x}}}k_i^{[1]} + L_{\text{x}}(x^{[1]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[1]}) + L_{\text{z}}z_i^{[1]}, L_{\text{u}}u\right) \quad \forall i = 1,\dots,s \end{aligned} \tag{3.14a}$$

$$\begin{aligned} 0 &= E_{21}k_i^{[1]} + E_{22}z_i^{[1]} - A_2(x_n^{[1]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[1]}) - B_2 u \\ &- C_2\phi\left(L_{\dot{\text{x}}}k_i^{[1]} + L_{\text{x}}(x^{[1]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[1]}) + L_{\text{z}}z_i^{[1]}, L_{\text{u}}u\right) \quad \forall i = 1,\dots,s \end{aligned} \tag{3.14b}$$

$$\begin{aligned} 0 &= E^{\text{LO}}k_i^{[2]} - A^{\text{LO}}(x_n^{[2]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[2]}) \\ &- f_{\text{LO}}(k_i^{[1]}, x_n^{[1]} + T_{\text{int}}\sum_{j=1}^{s} a_{ij}k_j^{[1]}, z_i^{[1]}, u) \quad \forall i = 1,\dots,s. \end{aligned} \tag{3.14c}$$

For notational convenience, we gather the integration variables as follows:

$$\begin{aligned} K^{[1]} &= \left({k_1^{[1]}}^\top, \dots, {k_s^{[1]}}^\top\right)^\top, \quad Z^{[1]} = \left({z_1^{[1]}}^\top, \dots, {z_s^{[1]}}^\top\right)^\top, \\ K^{[2]} &= \left({k_1^{[2]}}^\top, {z_1^{[2]}}^\top, \dots, {k_s^{[2]}}^\top, {z_s^{[2]}}^\top\right)^\top. \end{aligned}$$

Note that (3.14a) and (3.14b) do not depend on $K^{[2]}$. For that reason, we are able to solve first these equations for $K^{[1]}$ and Z up to some precision and subsequently solve equation (3.14c) for $K^{[2]}$. Using the above introduced notation, we are now able to rewrite (3.14a) and (3.14b) in the compact form

$$\begin{aligned}
&\mathbf{E}_1 K^{[1]} - \mathbf{D}_1 Z^{[1]} - \mathbf{A}_1 x_n^{[1]} - \mathbf{B}_1 u - \mathbf{C}_1 \Phi(\mathbf{L}_\mathrm{K} K^{[1]} + \mathbf{L}_\mathrm{x} x_n^{[1]} + \mathbf{L}_\mathrm{Z} Z^{[1]}, L_\mathrm{u} u) - \mathbf{c}_1 = 0 \\
&\mathbf{E}_2 Z^{[1]} - \mathbf{D}_2 K^{[1]} - \mathbf{A}_2 x_n^{[1]} - \mathbf{B}_2 u - \mathbf{C}_2 \Phi(\mathbf{L}_\mathrm{K} K^{[1]} + \mathbf{L}_\mathrm{x} x_n^{[1]} + \mathbf{L}_\mathrm{Z} Z^{[1]}, L_\mathrm{u} u) - \mathbf{c}_2 = 0.
\end{aligned} \tag{3.15}$$

Hereby the matrices $\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_1, \mathbf{B}_2, \mathbf{C}_1, \mathbf{C}_2, \mathbf{D}_1, \mathbf{D}_2, \mathbf{E}_1, \mathbf{E}_2$, are defined as follows

$$\begin{aligned}
&\mathbf{A}_1 = \begin{bmatrix} A_1 \\ \vdots \\ A_1 \end{bmatrix}, \ \mathbf{A}_2 = \begin{bmatrix} A_2 \\ \vdots \\ A_2 \end{bmatrix}, \ \mathbf{B}_1 = \begin{bmatrix} B_1 \\ \vdots \\ B_1 \end{bmatrix}, \ \mathbf{B}_2 = \begin{bmatrix} B_2 \\ \vdots \\ B_2 \end{bmatrix}, \\
&\mathbf{C}_1 = \mathbb{1}_s \otimes C_1, \quad \mathbf{C}_2 = \mathbb{1}_s \otimes C_2, \quad \mathbf{D}_1 = -\mathbb{1}_s \otimes E_{12}, \\
&\mathbf{D}_2 = T_\mathrm{int} A_\mathrm{but} \otimes A_2 - \mathbb{1}_s \otimes E_{21}, \\
&\mathbf{E}_1 = \mathbb{1}_s \otimes E_{11} - T_\mathrm{int} A_\mathrm{but} \otimes A_1, \quad \mathbf{E}_2 = \mathbb{1}_s \otimes E_{22},
\end{aligned} \tag{3.16}$$

where $\otimes$ is used to denote the Kronecker product. The matrices $\mathbf{L}_\mathrm{Z}, \mathbf{L}_\mathrm{K}, \mathbf{L}_\mathrm{x}$ and the vectors $\mathbf{c}_1, \mathbf{c}_2$ read as

$$\begin{aligned}
&\mathbf{L}_\mathrm{Z} = \mathbb{1}_s \otimes L_\mathrm{z}, \quad \mathbf{L}_\mathrm{K} = T_\mathrm{int} A_\mathrm{but} \otimes L_\mathrm{x} + \mathbb{1}_s \otimes L_{\dot{\mathrm{x}}}, \\
&\mathbf{L}_\mathrm{x} = \begin{bmatrix} L_\mathrm{x} \\ \vdots \\ L_\mathrm{x} \end{bmatrix}, \quad \mathbf{c}_1 = \begin{bmatrix} c_1 \\ \vdots \\ c_1 \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} c_2 \\ \vdots \\ c_2 \end{bmatrix}.
\end{aligned} \tag{3.17}$$

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

$$\Phi(\mathbf{y}, u) = \begin{bmatrix} \phi(y_1, L_\mathrm{u} u) \\ \vdots \\ \phi(y_s, L_\mathrm{u} u) \end{bmatrix} \quad \text{for} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_s \end{bmatrix} \in \mathbb{R}^{sn_\mathrm{in}}. \tag{3.18}$$

3.3.3 A Lifting-Condensing Technique for the IRK Equations

Now, we want to apply a lifting-condensing technique similar to the one suggested in [47]. We rewrite Equation (3.15) equivalently as

$$\mathbf{E}_1 K^{[1]} - \mathbf{D}_1 Z^{[1]} - \mathbf{A}_1 x_n^{[1]} - \mathbf{B}_1 u - \mathbf{C}_1 \mathbf{v} - \mathbf{c}_1 = 0, \tag{3.19a}$$

$$\mathbf{E}_2 Z^{[1]} - \mathbf{D}_2 K^{[1]} - \mathbf{A}_2 x_n^{[1]} - \mathbf{B}_2 u - \mathbf{C}_2 \mathbf{v} - \mathbf{c}_2 = 0, \tag{3.19b}$$

$$\mathbf{v} - \Phi(\mathbf{L}_\mathrm{x} x_n^{[1]} + \mathbf{L}_\mathrm{K} K^{[1]} + \mathbf{L}_\mathrm{Z} Z^{[1]},\, L_\mathrm{u} u) = 0, \tag{3.19c}$$

where we have introduced a new vector of variables $\mathbf{v} \in \mathbb{R}^{s n_\mathrm{out}}$ to lift the nonlinear terms in the IRK equations (3.15).
Observe that, since the matrices E_{11} and E_{22} are assumed to be invertible, $\mathbf{E}_2$ is always invertible and $\mathbf{E}_1$ is invertible for a sufficiently small integration step size [31]. Also notice that (3.19a) and (3.19b) are linear which enables us to rearrange these equations as

$$K^{[1]} = \mathbf{E}_1^{-1}(\mathbf{D}_1 Z^{[1]} + \mathbf{A}_1 x_n^{[1]} + \mathbf{B}_1 u + \mathbf{C}_1 \mathbf{v} + \mathbf{c}_1), \tag{3.20a}$$

$$Z^{[1]} = \mathbf{E}_2^{-1}(\mathbf{D}_2 K^{[1]} + \mathbf{A}_2 x_n^{[1]} + \mathbf{B}_2 u + \mathbf{C}_2 \mathbf{v} + \mathbf{c}_2). \tag{3.20b}$$

Now, we can plug (3.20a) into (3.20b) and obtain:

$$Z^{[1]} = \mathbf{E}_2^{-1}(\mathbf{D}_2 \mathbf{E}_1^{-1}(\mathbf{D}_1 Z^{[1]} + \mathbf{A}_1 x_n^{[1]} + \mathbf{B}_1 u + \mathbf{C}_1 \mathbf{v} + \mathbf{c}_1) + \mathbf{A}_2 x_n^{[1]} + \mathbf{B}_2 u + \mathbf{C}_2 \mathbf{v} + \mathbf{c}_2)$$

which is equivalent to

$$\begin{aligned}(\mathbb{1} - \mathbf{E}_2^{-1}\mathbf{D}_2\mathbf{E}_1^{-1}\mathbf{D}_1) Z^{[1]} \\ = \mathbf{E}_2^{-1}(\mathbf{D}_2 \mathbf{E}_1^{-1}(\mathbf{D}_1 Z^{[1]} + \mathbf{A}_1 x_n^{[1]} + \mathbf{B}_1 u + \mathbf{C}_1 \mathbf{v} + \mathbf{c}_1) + \mathbf{A}_2 x_n^{[1]} + \mathbf{B}_2 u + \mathbf{C}_2 \mathbf{v} + \mathbf{c}_2).\end{aligned}$$

We can use this to condense $Z^{[1]}$, i.e. determine a linear expression to compute it given the variables we want to keep in the structure exploiting integration scheme, namely $\mathbf{v}, x_n^{[1]}, u$. Thus, we get the following expression for $Z^{[1]}$

$$Z^{[1]} = \mathbf{Z}_\mathbf{v} \mathbf{v} + \mathbf{Z}_\mathrm{u} u + \mathbf{Z}_\mathrm{x} x_n^{[1]} + \mathbf{Z}_0, \tag{3.21}$$

where we introduced the matrices $\mathbf{Z}_\mathbf{v}, \mathbf{Z}_\mathrm{u}, \mathbf{Z}_\mathrm{x}, \mathbf{Z}_0$, which are computed as

$$\begin{aligned}
\mathbf{Q}_1 &= \left(\mathbb{1} - \left(\mathbf{E}_2^{-1}\mathbf{D}_2\right)\left(\mathbf{E}_1^{-1}\mathbf{D}_1\right)\right), \\
\mathbf{Z}_\mathbf{v} &= \mathbf{Q}_1^{-1}\left(\left(\mathbf{E}_2^{-1}\mathbf{D}_2\right)\left(\mathbf{E}_1^{-1}\mathbf{C}_1\right) + \left(\mathbf{E}_2^{-1}\mathbf{C}_2\right)\right), \\
\mathbf{Z}_\mathrm{u} &= \mathbf{Q}_1^{-1}\left(\left(\mathbf{E}_2^{-1}\mathbf{D}_2\right)\left(\mathbf{E}_1^{-1}\mathbf{B}_1\right) + \left(\mathbf{E}_2^{-1}\mathbf{B}_2\right)\right), \\
\mathbf{Z}_\mathrm{x} &= \mathbf{Q}_1^{-1}\left(\left(\mathbf{E}_2^{-1}\mathbf{D}_2\right)\left(\mathbf{E}_1^{-1}\mathbf{A}_1\right) + \left(\mathbf{E}_2^{-1}\mathbf{A}_2\right)\right), \\
\mathbf{Z}_0 &= \mathbf{Q}_1^{-1}\left(\left(\mathbf{E}_2^{-1}\mathbf{D}_2\right)\left(\mathbf{E}_1^{-1}\mathbf{c}_1\right) + \left(\mathbf{E}_2^{-1}\mathbf{c}_2\right)\right).
\end{aligned} \tag{3.22}$$

Note that for computing these quantities, the matrices $\mathbf{E}_1, \mathbf{E}_2, \mathbf{Q}_1$ have to be factorized and the expressions marked with brackets in (3.22) can be computed offline.
We proceed by condensing $K^{[1]}$ similarly, plugging (3.22) into (3.20a) to obtain the expression

$$K^{[1]} = \mathbf{K}_\mathbf{v}\mathbf{v} + \mathbf{K}_\mathrm{u}u + \mathbf{K}_\mathrm{x}x_n^{[1]} + \mathbf{K}_0, \tag{3.23}$$

where

$$\begin{aligned}
\mathbf{K}_\mathbf{v} &= \left(\mathbf{E}_1^{-1}\mathbf{D}_1\right)\mathbf{Z}_\mathbf{v} + \left(\mathbf{E}_1^{-1}\mathbf{C}_1\right), \\
\mathbf{K}_\mathrm{u} &= \left(\mathbf{E}_1^{-1}\mathbf{D}_1\right)\mathbf{Z}_\mathrm{u} + \left(\mathbf{E}_1^{-1}\mathbf{B}_1\right), \\
\mathbf{K}_\mathrm{x} &= \left(\mathbf{E}_1^{-1}\mathbf{D}_1\right)\mathbf{Z}_\mathrm{x} + \left(\mathbf{E}_1^{-1}\mathbf{A}_1\right), \\
\mathbf{K}_0 &= \left(\mathbf{E}_1^{-1}\mathbf{D}_1\right)\mathbf{Z}_0 + \left(\mathbf{E}_1^{-1}\mathbf{c}_1\right).
\end{aligned} \tag{3.24}$$

One can observe that all expressions in brackets have already been computed in (3.22) and can be reused.

Remark 3.10 (Regularity of $\mathbf{Q}_1$) In the precomputation phase, the simulation algorithm should check if the matrix $\left(\mathbb{1} - \mathbf{E}_2^{-1}\mathbf{D}_2\mathbf{E}_1^{-1}\mathbf{D}_1\right)$ is invertible. This is most often the case because $\mathbb{1}$ is invertible and the matrix to be subtracted is not of full rank (an upper bound is $s\min\{n_{\mathrm{z}_1}, n_{\mathrm{x}_1}\}$). However, if the matrix $\mathbf{Q}_1$ is not invertible, one can change the Runge-Kutta formula or the integration step size, since both $\mathbf{E}_1$ and $\mathbf{D}_2$ are depending on these quantities.

Alternatively, the matrices $\mathbf{Z}_\mathbf{v}, \mathbf{Z}_\mathrm{u}, \mathbf{Z}_\mathrm{x}, \mathbf{Z}_0, \mathbf{K}_\mathbf{v}, \mathbf{K}_\mathrm{u}, \mathbf{K}_\mathrm{x}, \mathbf{K}_0$ could be obtained by plugging (3.20b) into (3.20a). However, this is most often computationally more expensive, as instead of factorizing $\mathbf{Q}_1 \in \mathbb{R}^{sn_{\mathrm{z}_1} \times sn_{\mathrm{z}_1}}$, the matrix $(\mathbb{1} - \mathbf{E}_1^{-1}\mathbf{D}_1\mathbf{E}_2^{-1}\mathbf{D}_2)^{-1} \in \mathbb{R}^{sn_{\mathrm{x}_1} \times sn_{\mathrm{x}_1}}$ has to be factorized, which is usually of higher dimension.
Finally, we derive an expression for the input argument of the gathered nonlinearity

function Φ, which reads as

$$\begin{aligned}\mathbf{y} &= \mathbf{L}_{\mathrm{x}} x_n^{[1]} + \mathbf{L}_{\mathrm{K}} K^{[1]} + \mathbf{L}_{\mathrm{Z}} Z^{[1]} \\ &= \mathbf{L}_{\mathrm{x}} x_n^{[1]} + \mathbf{L}_{\mathrm{K}}(\mathbf{K}_{\mathbf{v}}\mathbf{v} + \mathbf{K}_{\mathrm{u}} u + \mathbf{K}_{\mathrm{x}} x_n^{[1]} + \mathbf{K}_0) + \mathbf{L}_{\mathrm{Z}}(\mathbf{Z}_{\mathbf{v}}\mathbf{v} + \mathbf{Z}_{\mathrm{u}} u + \mathbf{Z}_{\mathrm{x}} x_n^{[1]} + \mathbf{Z}_0) \\ &= \mathbf{Y}_{\mathrm{x}} x_n^{[1]} + \mathbf{Y}_{\mathrm{u}} u + \mathbf{Y}_{\mathbf{v}}\mathbf{v} + \mathbf{Y}_0,\end{aligned} \tag{3.25}$$

where the matrices $\mathbf{Y}_{\mathbf{v}}, \mathbf{Y}_{\mathrm{u}}, \mathbf{Y}_{\mathrm{x}}, \mathbf{Y}_0$ are introduced as

$$\begin{aligned}\mathbf{Y}_{\mathrm{x}} &= \mathbf{L}_{\mathrm{x}} + \mathbf{L}_{\mathrm{K}}\mathbf{K}_{\mathrm{x}} + \mathbf{L}_{\mathrm{Z}}\mathbf{Z}_{\mathrm{x}}, \\ \mathbf{Y}_{\mathrm{u}} &= \mathbf{L}_{\mathrm{K}}\mathbf{K}_{\mathrm{u}} + \mathbf{L}_{\mathrm{Z}}\mathbf{Z}_{\mathrm{u}}, \\ \mathbf{Y}_{\mathbf{v}} &= \mathbf{L}_{\mathrm{K}}\mathbf{K}_{\mathbf{v}} + \mathbf{L}_{\mathrm{Z}}\mathbf{Z}_{\mathbf{v}}, \\ \mathbf{Y}_0 &= \mathbf{L}_{\mathrm{K}}\mathbf{K}_0 + \mathbf{L}_{\mathrm{Z}}\mathbf{Z}_0.\end{aligned} \tag{3.26}$$

All these bold matrices can be precomputed offline, but only after the choice of the integrator options, i.e. Butcher tableau and 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:

$$\mathbf{r}(\mathbf{v}, x_n^{[1]}, u) = \mathbf{v} - \Phi(\mathbf{Y}_{\mathrm{x}} x_n^{[1]} + \mathbf{Y}_{\mathrm{u}} u + \mathbf{Y}_{\mathbf{v}}\mathbf{v} + \mathbf{Y}_0,\, L_{\mathrm{u}} u) = 0. \tag{3.27}$$

A full-step Newton iteration then reads as:

$$\begin{aligned}\mathbf{v} &\leftarrow \mathbf{v} + \Delta\mathbf{v}, \\ \text{where} \quad \Delta\mathbf{v} &= -\left(\frac{\partial \mathbf{r}}{\partial \mathbf{v}}(\mathbf{v}, x_n^{[1]}, u)\right)^{-1} \mathbf{r}(\mathbf{v}, x_n^{[1]}, u).\end{aligned} \tag{3.28}$$

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

$$\begin{aligned}\frac{\partial \mathbf{r}}{\partial \mathbf{v}}(\cdot) &= \mathbb{1} - \frac{\partial \Phi}{\partial \mathbf{y}}(\cdot)\, \mathbf{Y}_{\mathbf{v}} \\ \frac{\partial \mathbf{r}}{\partial (x_n^{[1]}, u)}(\cdot) &= -\frac{\partial \Phi}{\partial \mathbf{y}}(\cdot) \begin{bmatrix} \mathbf{Y}_{\mathrm{x}} & \mathbf{Y}_{\mathrm{u}} \end{bmatrix} - \begin{bmatrix} \mathbb{0} & \frac{\partial \Phi}{\partial \hat{u}}(\cdot) L_{\mathrm{u}} \end{bmatrix},\end{aligned} \tag{3.29}$$

where

$$\frac{\partial \Phi}{\partial \mathbf{y}}(\mathbf{y}, u) = \operatorname{diag}\left(\frac{\partial \phi}{\partial y}(y_i, u)\right)_{i=1,\ldots,s}. \tag{3.30}$$

Within the `acados` implementation of GNSF-IRK, ϕ, $\frac{\partial \phi}{\partial y}$ are evaluated by a `C` function. The model functions are typically code generated using `CasADi`.

3.3.4 Simulation of the Linear Output System

Now, we can solve the IRK equations (2.18a) corresponding to the linear output system (3.12b). This can be written as

$$G_2(w_n, K^{[1]}, Z^{[1]}, K^{[2]}) = 0, \tag{3.31}$$

where the function G_2 is defined as

$$\begin{aligned} G_2(w_n, K^{[1]}, Z^{[1]}, K^{[2]}) := \Bigg[E^{\mathrm{LO}} \begin{bmatrix} k_i^{[2]} \\ z_i^{[2]} \end{bmatrix} - A^{\mathrm{LO}}(x_n^{[2]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[2]}) \\ - f_{\mathrm{LO}}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u) \Bigg]_{i=1,\dots,s}. \end{aligned} \tag{3.32}$$

Because (3.31) 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 a 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

$$M_2 = \mathbb{1}_q \otimes E^{\mathrm{LO}} - T_{\mathrm{int}} A_{\mathrm{but}} \otimes \begin{bmatrix} A^{\mathrm{LO}} & \mathbb{0}_{(n_{\mathrm{x}_2}+n_{\mathrm{z}_2}) \times n_{\mathrm{z}_2}} \end{bmatrix}. \tag{3.33}$$

The final state of the integrator is given by

$$x_{n+1} = x_n + T_{\mathrm{int}} \sum_{j=1}^{s} b_j k_j, \tag{3.34}$$

where $k_j = \left({k_j^{[1]}}^\top, {k_j^{[2]}}^\top \right)^\top$.
The GNSF-IRK scheme is presented as pseudo code in Algorithm 3, not including any sensitivity propagation.

3.3.5 First Order Sensitivities

This subsection aims at giving a detailed derivation for efficient first order sensitivity propagation within the GNSF-IRK scheme.

Algorithm 3: GNSF-IRK scheme - simple simulation

Input: x_0, u, T_{int}, $n_{\text{steps}}, n_{\text{newton}}$, precomputed matrices given a Butcher tableau $\mathbf{Y}_{\text{x}}, \mathbf{Y}_{\text{u}}, \mathbf{Y}_{\mathbf{v}}, \mathbf{Y}_0$, $\mathbf{K}_{\text{x}}, \mathbf{K}_{\text{u}}, \mathbf{K}_{\mathbf{v}}, \mathbf{K}_0, \mathbf{Z}_{\text{x}}, \mathbf{Z}_{\text{u}}, \mathbf{Z}_{\mathbf{v}}, \mathbf{Z}_0$, factorization of M_2
Initialize $\mathbf{v}$ with $\mathbb{0}$ or previous solution
$\mathbf{y}^{\text{u}} \leftarrow \mathbf{Y}_0 + \mathbf{Y}_{\text{u}} u$ ▷ Compute (3.26) successively, $K^{[1]}, Z$ similarly
for $s = 1 : n_{\text{steps}}$ **do**
    $\mathbf{y}^{\text{s}} \leftarrow \mathbf{y}^{\text{u}} + \mathbf{Y}_{\text{x}} x_{s-1}^{[1]}$

`Simulate NSF type system` (3.12a)
    for $i_{\text{newton}} = 1 : n_{\text{newton}}$ **do**
        $\mathbf{y} \leftarrow \mathbf{y}^{\text{s}} + \mathbf{Y}_{\mathbf{v}} \mathbf{v}$
        Evaluate $\phi(y_i, \hat{u}), \frac{\partial \phi}{\partial y}(y_i, \hat{u}) \quad \forall i = 1, \ldots, s$ ▷ using an AD tool
        Compute residual and Jacobian $\mathbf{r}$, $\frac{\partial \mathbf{r}}{\partial \mathbf{v}}$ ▷ via (3.27), (3.29), (3.30)
        Solve linear system for $\Delta \mathbf{v}$ ▷ via (3.28)
        Apply Newton step $\mathbf{v} \leftarrow \mathbf{v} + \Delta \mathbf{v}$ ▷ via (3.28)
    end for
    Obtain $K^{[1]}, Z^{[1]}$ ▷ via (3.23), (3.21)

`Simulate LOS` (3.12b)
    $G_2{}^{\text{val}} \leftarrow G_2(w_n, K^{[1]}, Z^{[1]}, 0)$ ▷ via (3.32)
    Solve linear system $M_2 K^{[2]} = -G_2{}^{\text{val}}$ ▷ M_2 factorization precomputed

`Obtain simulation result`
    $x_s = x_{s-1} + T_{\text{int}} \sum_{j=1}^{q} b_j k_j$
end for

Forward Sensitivities

In this paragraph we establish an efficient way to compute the forward sensitivities $\frac{\partial x_{n+1}}{\partial w_n} = \left[\frac{\partial x_{n+1}}{\partial x_n}, \frac{\partial x_{n+1}}{\partial u_n}\right]$. Let us start from Equation (2.9). We observe that

$$\begin{aligned} \frac{\partial x_{n+1}(x_n, u_n)}{\partial w_n} &= \frac{\partial \Psi}{\partial w_n} + \frac{\partial \Psi}{\partial K} \frac{\partial K}{\partial w_n} \\ &= \mathbb{1}_{n_{\text{x}} \times (n_{\text{x}} + n_{\text{u}})} + T_{\text{int}} \sum_{j=1}^{s} b_j \frac{\partial k_j}{\partial w_n}, \end{aligned} \tag{3.35}$$

where $\frac{\partial k_j}{\partial w_n} = \begin{bmatrix} \frac{\partial k_j^{[1]}}{\partial w_n} \\ \frac{\partial k_j^{[2]}}{\partial w_n} \end{bmatrix}$.

We first derive expressions for first order derivatives of the integration variables corresponding to the NSF type system (3.12a). For $K^{[1]}$ and $Z^{[1]}$, we use (3.23) and

(3.21) respectively to directly obtain

$$\frac{\partial K^{[1]}}{\partial w_n} = \left[\frac{\partial K^{[1]}}{\partial x_n^{[1]}}, \frac{\partial K^{[1]}}{\partial x_n^{[2]}}, \frac{\partial K^{[1]}}{\partial u}\right] = \left[\mathbf{K}_{\mathrm{x}} + \mathbf{K}_{\mathbf{v}}\frac{\partial \mathbf{v}}{\partial x_n^{[1]}}, \quad \mathbb{0}, \quad \mathbf{K}_{\mathrm{u}} + \mathbf{K}_{\mathbf{v}}\frac{\partial \mathbf{v}}{\partial u}\right], \tag{3.36a}$$

$$\frac{\partial Z^{[1]}}{\partial w_n} = \left[\frac{\partial Z^{[1]}}{\partial x_n^{[1]}}, \frac{\partial Z^{[1]}}{\partial x_n^{[2]}}, \frac{\partial Z^{[1]}}{\partial u}\right] = \left[\mathbf{Z}_{\mathrm{x}} + \mathbf{Z}_{\mathbf{v}}\frac{\partial \mathbf{v}}{\partial x_n^{[1]}}, \quad \mathbb{0}, \quad \mathbf{Z}_{\mathrm{u}} + \mathbf{Z}_{\mathbf{v}}\frac{\partial \mathbf{v}}{\partial u}\right]. \tag{3.36b}$$

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.

The sensitivities $\frac{\partial \mathbf{v}}{\partial x^{[1]},u}$ can be obtained by applying the implicit function theorem on Equation (3.27), yielding the following system of linear equations

$$\frac{\partial \mathbf{r}}{\partial \mathbf{v}} \cdot \frac{\partial \mathbf{v}}{\partial (x^{[1]}, u)} = -\left(\frac{\partial \mathbf{r}}{\partial (x^{[1]}, u)}\right). \tag{3.37}$$

Here, the matrix to be factorized is the same as in the simulation part, see Equations (3.28) and (3.29).

Now looking at the $K^{[2]}$ variables, we apply the implicit function theorem to Equation (3.31) and obtain

$$\frac{\partial K^{[2]}}{\partial w_n} = -M_2^{-1}\left[\frac{\partial G_2}{\partial w_n} + \frac{\partial G_2}{\partial K^{[1]}}\frac{\partial K^{[1]}}{\partial w_n} + \frac{\partial G_2}{\partial Z^{[1]}}\frac{\partial Z^{[1]}}{\partial w_n}\right]. \tag{3.38}$$

Here, the needed partial derivatives of the G_2 are constructed as follows:

$$\begin{aligned}
\frac{\partial G_2}{\partial x^{[1]}} &= \left[-\frac{\partial f_{\mathrm{LO}}}{\partial x^{[1]}}(k_i^{[1]}, x_n^{[1]} T_{\mathrm{int}} + \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u)\right]_{i=1,\ldots,s} \qquad \frac{\partial G_2}{\partial x^{[2]}} = -\begin{bmatrix} A^{\mathrm{LO}} \\ \ldots \\ A^{\mathrm{LO}} \end{bmatrix} \\
\frac{\partial G_2}{\partial u} &= \left[-\frac{\partial f_{\mathrm{LO}}}{\partial u}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u)\right]_{i=1,\ldots,s} \\
\frac{\partial G_2}{\partial K^{[1]}} &= \Bigg[-T_{\mathrm{int}} a_{ij} \frac{\partial f_{\mathrm{LO}}}{\partial x^{[1]}}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u) \\
&\qquad -\delta_{ij} \frac{\partial f_{\mathrm{LO}}}{\partial \dot{x}^{[1]}}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u)\Bigg]_{i,j=1,\ldots,s} \\
\frac{\partial G_2}{\partial Z^{[1]}} &= \mathrm{diag}\left(-\frac{\partial f_{\mathrm{LO}}}{\partial z}(k_i^{[1]}, x_n^{[1]} + T_{\mathrm{int}} \sum_{j=1}^{s} a_{ij} k_j^{[1]}, z_i^{[1]}, u)\right)_{i=1,\ldots,s}.
\end{aligned} \tag{3.39}$$

Observe that the derivatives $\frac{\partial K^{[2]}}{\partial x_n^{[2]}}$ are constant and can be precomputed. Furthermore, in an efficient implementation $\frac{\partial G_2}{\partial Z^{[1]}}$ is stored as a compressed block matrix.

Adjoint Sensitivities

In this paragraph, we derive an efficient way to propagate adjoint sensitivities within the GNSF-IRK scheme. We use the general derivation in Section 2.3.2. Therefore, the integration scheme has to be regarded in the general form (2.8). In Table 3, the notational counterparts of the GNSF-IRK scheme in the context of Section 2.3.2 is clarified.

	Context Section 2.3.2	Context GNSF-IRK
Integration Variables	K	$\mathbf{v}$
Integration Equations	$G(x_n, K_n, u)$	$\mathbf{r}(x_n, \mathbf{v}, u)$
Output Function	$\Psi(x_n, K_n, u)$	$\begin{bmatrix} \Psi_1(x_n, \mathbf{v}, u) \\ \Psi_2(x_n, \mathbf{v}, u) \end{bmatrix}$

Table 3: GNSF-IRK in the context of a general integration method

Hereby, we revise that $K^{[1]}, K^{[2]}$ are obtained by

$$K^{[1]} = \mathbf{K}_\mathbf{v}\mathbf{v} + \mathbf{K}_\mathrm{u}u + \mathbf{K}_\mathrm{x}x_n^{[1]} + \mathbf{K}_0, \tag{3.40a}$$

$$K^{[2]} = -M_2^{-1}G_2(w_n, K^{[1]}, Z^{[1]}, 0). \tag{3.40b}$$

We formally define the output functions of the scheme Ψ_1, Ψ_2 as

$$\begin{bmatrix} \Psi_1(x_n, \mathbf{v}, u) \\ \Psi_2(x_n, \mathbf{v}, u) \end{bmatrix} = \begin{bmatrix} x_n^{[1]} + T_\mathrm{int}\sum_{i=1}^s b_i k_i^{[1]}(x_n, \mathbf{v}, u) \\ x_n^{[2]} + T_\mathrm{int}\sum_{i=1}^s b_i k_i^{[2]}(x_n, \mathbf{v}, u) \end{bmatrix}. \tag{3.41}$$

The required derivatives of Ψ_1 can be obtained by

$$\begin{aligned} \frac{\partial \Psi_1}{\partial x_n^{[1]}} &= \mathbb{1}_{n_{\mathrm{x}_1}} + T_\mathrm{int}\sum_{i=1}^s b_i \mathbf{K}_\mathrm{x}^i, \quad \frac{\partial \Psi_1}{\partial x_n^{[2]}} = \mathbb{0}, \\ \frac{\partial \Psi_1}{\partial \mathbf{v}} &= T_\mathrm{int}\sum_{i=1}^s b_i \mathbf{K}_\mathbf{v}{}^i, \quad \frac{\partial \Psi_1}{\partial u} = T_\mathrm{int}\sum_{i=1}^s b_i \mathbf{K}_\mathrm{u}^i, \end{aligned} \tag{3.42}$$

where we divide the matrices $\mathbf{K}_\mathbf{v}, \mathbf{K}_\mathrm{u}, \mathbf{K}_\mathrm{x}$ naturally as

$$\mathbf{K}_\mathbf{v} = \begin{bmatrix} \mathbf{K}_\mathbf{v}{}^1 \\ \vdots \\ \mathbf{K}_\mathbf{v}{}^s \end{bmatrix}, \quad \mathbf{K}_\mathrm{u} = \begin{bmatrix} \mathbf{K}_\mathrm{u}^1 \\ \vdots \\ \mathbf{K}_\mathrm{u}^s \end{bmatrix}, \quad \mathbf{K}_\mathrm{x} = \begin{bmatrix} \mathbf{K}_\mathrm{x}^1 \\ \vdots \\ \mathbf{K}_\mathrm{x}^s \end{bmatrix}. \tag{3.43}$$

Furthermore, the derivatives of Ψ_2 are given as

$$\begin{aligned}
\frac{\partial \Psi_2}{\partial x_n^{[1]}} &= T_{\text{int}} \sum_{i=1}^{s} b_i \frac{\partial k_i^{[2]}}{\partial x_n^{[1]}}, \quad \frac{\partial \Psi_2}{\partial x_n^{[2]}} = \mathbb{1}_{n_{\text{x}_2}} + T_{\text{int}} \sum_{i=1}^{s} b_i \frac{\partial k_i^{[2]}}{\partial x_n^{[2]}}, \\
\frac{\partial \Psi_2}{\partial \mathbf{v}} &= T_{\text{int}} \sum_{i=1}^{s} b_i \frac{\partial k_i^{[2]}}{\partial \mathbf{v}}, \quad \frac{\partial \Psi_2}{\partial u} = T_{\text{int}} \sum_{i=1}^{s} b_i \frac{\partial k_i^{[2]}}{\partial u},
\end{aligned} \tag{3.44}$$

where the derivatives of $K^{[2]}$ can be calculated by

$$\begin{aligned}
\frac{\partial K^{[2]}}{\partial x_n^{[1]}} &= -M_2^{-1} \left(\frac{\partial G_2}{\partial x_n^{[1]}} + \frac{\partial G_2}{\partial K^{[1]}} \mathbf{K}_{\text{x}} + \frac{\partial G_2}{\partial Z^{[1]}} \mathbf{Z}_{\text{x}} \right), \\
\frac{\partial K^{[2]}}{\partial u} &= -M_2^{-1} \left(\frac{\partial G_2}{\partial u} + \frac{\partial G_2}{\partial K^{[1]}} \mathbf{K}_{\text{u}} + \frac{\partial G_2}{\partial Z^{[1]}} \mathbf{Z}_{\text{u}} \right), \\
\frac{\partial K^{[2]}}{\partial \mathbf{v}} &= -M_2^{-1} \left(\frac{\partial G_2}{\partial \mathbf{K^{[1]}}} \mathbf{K}_{\mathbf{v}} + \frac{\partial G_2}{\partial Z^{[1]}} \mathbf{Z}_{\mathbf{v}} \right).
\end{aligned} \tag{3.45}$$

For the derivatives of G_2, we refer to (3.39), and mention again that $\frac{\partial K^{[2]}}{\partial x_n^{[2]}}$ is constant and can be precomputed.

3.3.6 Sensitivity Propagation for the Algebraic Variables

In this section, we derive a way to efficiently compute the sensitivities of the algebraic variables $\frac{\partial z_n}{\partial w_n}$ within the GNSF-IRK scheme. It exploits the GNSF structure and is therefore more efficient than the general approach described in Section 2.3.4.
The following derivation is very similar to the one in the previous sections, because the system of equations has a very similar structure but is of dimension n_{out} instead of sn_{out}. It shall still be described here for the sake of completeness.
Let us first regard the $z^{[1]}$ component. One observes that it does not depend on the linear output type system, i.e. $\frac{\partial z_n^{[1]}}{\partial x_n^{[2]}} = \mathbb{0}$. Thus, we first just regard (3.12a). In this context the function corresponding to f^{impl} in Section 2.3.4, is

$$\begin{aligned}
0 &= f^{\text{impl}[1]}(\dot{x}^{[1]}, x^{[1]}, z^{[1]}, u) \\
&= E \begin{bmatrix} \dot{x}^{[1]} \\ z^{[1]} \end{bmatrix} - A x^{[1]} - B u - C \phi(L_{\dot{\text{x}}} \dot{x}^{[1]} + L_{\text{x}} x^{[1]} + L_{\text{z}} z^{[1]}, L_{\text{u}} u) - c.
\end{aligned} \tag{3.46}$$

The system of equations (3.46) can be split as

$$\begin{aligned} 0 &= E_{11}\dot{x}_n^{[1]} + E_{12}z_n^{[1]} - A_1 x_n^{[1]} - B_1 u - C_1 \phi\left(L_{\dot{\mathrm{x}}}\dot{x}_n^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z_n^{[1]}, L_{\mathrm{u}}u\right) - c_1 \\ 0 &= E_{21}\dot{x}_n^{[1]} + E_{22}z_n^{[1]} - A_2 x_n^{[1]} - B_2 u - C_2 \phi\left(L_{\dot{\mathrm{x}}}\dot{x}_n^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z_n^{[1]}, L_{\mathrm{u}}u\right) - c_2. \end{aligned} \tag{3.47}$$

Let us apply a lifting-condensing technique equivalent to the one we applied to the integration equations in the previous sections. We introduce the variable $\nu_0 \in \mathbb{R}^{n_{\mathrm{out}}}$ and use the fact that E_{11} and E_{22} are invertible. The system (3.47) is equivalent to

$$\dot{x}_n^{[1]} = (E_{11})^{-1}(A_1 x_n^{[1]} + B_1 u + C_1 \nu_0 + c_1 - E_{12} z_n^{[1]}), \tag{3.48a}$$

$$z_n^{[1]} = (E_{22})^{-1}(A_2 x_n^{[1]} + B_2 u + C_2 \nu_0 + c_2 - E_{21} \dot{x}_n^{[1]}), \tag{3.48b}$$

$$\nu_0 = \phi\left(L_{\dot{\mathrm{x}}}\dot{x}_n^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z_n^{[1]}, L_{\mathrm{u}}u\right). \tag{3.48c}$$

Plugging now (3.48a) into (3.48b) gives us

$$\begin{aligned} &(\mathbb{1} - E_{22}^{-1}E_{21}E_{11}^{-1}E_{12})z_n^{[1]} = \\ &\quad E_{22}^{-1}(A_2 x_n^{[1]} + B_2 u + C_2\nu_0 + c_2 - E_{21}E_{11}^{-1}(A_1 x_n^{[1]} + B_1 u + C_1\nu_0 + c_1)). \end{aligned} \tag{3.49}$$

Thus, we can write

$$z_n^{[1]} = {Z_{\mathrm{x}}}^0 x_n^{[1]} + {Z_{\mathrm{u}}}^0 u + {Z_{\mathrm{f}}}^0 \nu_0 + {Z_0}^0 \tag{3.50}$$

with

$$\begin{aligned} Q_1 &= (\mathbb{1} - E_{22}^{-1}E_{21}E_{11}^{-1}E_{12}), \\ {Z_{\mathrm{x}}}^0 &= Q_1^{-1}((E_{22}^{-1}E_{21})(E_{11}^{-1}A_1) + E_{22}^{-1}A_2), \\ {Z_{\mathrm{u}}}^0 &= Q_1^{-1}((E_{22}^{-1}E_{21})(E_{11}^{-1}B_1) + E_{22}^{-1}B_2), \\ {Z_{\mathrm{f}}}^0 &= Q_1^{-1}((E_{22}^{-1}E_{21})(E_{11}^{-1}C_1) + E_{22}^{-1}C_2), \\ {Z_0}^0 &= Q_1^{-1}((E_{22}^{-1}E_{21})(E_{11}^{-1}c_1) + E_{22}^{-1}c_2). \end{aligned} \tag{3.51}$$

By plugging (3.50) into (3.48a), we obtain

$$\dot{x}_n^{[1]} = {K_{\mathrm{x}}}^0 x_n^{[1]} + {K_{\mathrm{u}}}^0 u + {K_{\mathrm{f}}}^0 \nu_0 + {K_0}^0, \tag{3.52}$$

where the following matrices are introduced:

$$\begin{aligned} {K_{\mathrm{x}}}^0 &= E_{11}^{-1}(A_1 - E_{12}{Z_{\mathrm{x}}}^0), \\ {K_{\mathrm{u}}}^0 &= E_{11}^{-1}(B_1 - E_{12}{Z_{\mathrm{u}}}^0), \\ {K_{\mathrm{f}}}^0 &= E_{11}^{-1}(C_1 - E_{12}{Z_{\mathrm{f}}}^0), \\ {K_0}^0 &= E_{11}^{-1}(c_1 - E_{12}{Z_0}^0). \end{aligned} \tag{3.53}$$

Let us now rewrite Equation (3.48c) introducing the residual function r as

$$0 = r(\nu_0, x_n^{[1]}, u) = \nu_0 - \phi\left({Y_{\mathrm{x}}}^0 x_n^{[1]} + {Y_{\mathrm{u}}}^0 u + {Y_{\mathrm{f}}}^0 \nu_0 + {Y_0}^0, L_{\mathrm{u}} u\right), \tag{3.54}$$

where the following matrices are introduced

$$\begin{aligned} {Y_{\mathrm{x}}}^0 &= L_{\dot{\mathrm{x}}}{K_{\mathrm{x}}}^0 + L_{\mathrm{z}}{Z_{\mathrm{x}}}^0 + L_{\mathrm{x}}, \\ {Y_{\mathrm{u}}}^0 &= L_{\dot{\mathrm{x}}}{K_{\mathrm{u}}}^0 + L_{\mathrm{z}}{Z_{\mathrm{u}}}^0, \\ {Y_{\mathrm{f}}}^0 &= L_{\dot{\mathrm{x}}}{K_{\mathrm{f}}}^0 + L_{\mathrm{z}}{Z_{\mathrm{f}}}^0, \\ {Y_0}^0 &= L_{\dot{\mathrm{x}}}{K_0}^0 + L_{\mathrm{z}}{Z_0}^0. \end{aligned} \tag{3.55}$$

Differentiating (3.50) w.r.t. $\left(x_n^{[1]}, u\right)$, we obtain

$$\frac{\partial z_n^{[1]}}{\partial\left(x_n^{[1]}, u\right)} = \begin{bmatrix} {Z_{\mathrm{x}}}^0 & {Z_{\mathrm{u}}}^0 \end{bmatrix} + {Z_{\mathrm{f}}}^0 \frac{\partial \nu_0}{\partial\left(x_n^{[1]}, u\right)}. \tag{3.56}$$

We obtain an expression for $\frac{\partial \nu_0}{\partial\left(x_n^{[1]}, u\right)}$ by applying the implicit function theorem on (3.54)

$$\frac{\partial \nu_0}{\partial\left(x_n^{[1]}, u\right)} = -\frac{\partial r}{\partial \nu_0}^{-1} \frac{\partial r}{\partial\left(x_n^{[1]}, u\right)}. \tag{3.57}$$

Hereby, the derivatives of r are given by

$$\begin{aligned} \frac{\partial r}{\partial\left(x_n^{[1]}, u\right)} &= -\frac{\partial \phi}{\partial y}\begin{bmatrix} {Y_{\mathrm{x}}}^0 & {Y_{\mathrm{u}}}^0 \end{bmatrix} - \begin{bmatrix} \mathbb{0} & \frac{\partial \phi}{\partial \hat{u}} L_{\mathrm{u}} \end{bmatrix}, \\ \frac{\partial r}{\partial \nu_0} &= \mathbb{1} - \frac{\partial \phi}{\partial y}{Y_{\mathrm{f}}}^0. \end{aligned} \tag{3.58}$$

Note that $\dot{x}_n^{[1]}, z_n$ are still obtained as suggested in Section 2.3.4 by an interpolation formula.

Let us now regard the sensitivities of $z^{[2]}$. One observes that $\dot{x}_n^{[2]}, z_n^{[2]}$ is given by

$$\begin{bmatrix} \dot{x}_n^{[2]} \\ z_n^{[2]} \end{bmatrix} = \left(E^{\mathrm{LO}}\right)^{-1} \left(A^{\mathrm{LO}} x_n^{[2]} + f_{\mathrm{LO}}(\dot{x}_n^{[1]}, x^{[1]}, z_n^{[1]}, u)\right). \tag{3.59}$$

Applying the implicit function theorem to this equation one obtains

$$\frac{\mathrm{d}\left(\dot{x}_n^{[2]}, z_n^{[2]}\right)}{\mathrm{d}\left(x_n^{[1]}, u^{[2]}\right)} = \left(E^{\mathrm{LO}}\right)^{-1} \left(\frac{\partial f_{\mathrm{LO}}}{\partial\left(x^{[1]}, u\right)} + \frac{\partial f_{\mathrm{LO}}}{\partial z} \frac{\mathrm{d}z_n^{[1]}}{\mathrm{d}\left(x^{[1]}, u\right)} + \frac{\partial f_{\mathrm{LO}}}{\partial \dot{x}^{[1]}} \frac{\mathrm{d}\dot{x}_n^{[1]}}{\mathrm{d}\left(x^{[1]}, u\right)}\right). \tag{3.60}$$

Note that the matrix E^{LO} can be factorized offline and the sensitivities of $\dot{x}_n^{[1]}, z^{[1]}$ were derived in the first part of this section.
The derivative with respect to $x_n^{[2]}$ is constant and can be precomputed as

$$\frac{\mathrm{d}\left(\dot{x}_n^{[2]}, z_n^{[2]}\right)}{\mathrm{d}x_n^{[2]}} = \left(E^{\mathrm{LO}}\right)^{-1} A^{\mathrm{LO}}. \tag{3.61}$$

3.4 Discussion on GNSF-IRK

In this section, we want to discuss some further aspects of the proposed GNSF-IRK scheme, like design choices and the context of state-of-the-art algorithms.

3.4.1 Differences from other Dynamic System Exploiting IRK Schemes

A main difference of the derivation in Section 3.3 compared to the one in [47] is that the lifting-condensing technique in [47] is derived for a collocation method, whereas we take the more general approach of an IRK method.
Additionally, we use a DAE formulation compared to the ODE formulation in (3.9), which gives us an additional subsystem for the NSF type system, see Equation (3.15). The LOS formulation in (3.12b) is a generalization of the original one in (3.8c) because it supports the dependence on previously determined state derivatives, i.e. $\dot{x}^{[1]}$ and the matrix in front of the unknowns does not have to be $\mathbb{1}$.

3.4.2 Complexity Comparison

In order to compare the computational complexity of a standard IRK and the proposed GNSF-IRK scheme, we consider their Newton iterations and their complexity ratio, like it was done in [47].

Recall that the size of the Newton matrix that has to be factorized online is $s(n_\mathrm{x}+n_\mathrm{z})$ for the standard IRK and sn_out for the GNSF-IRK scheme. Asymptotically, for a very high number of stages s and a GNSF model with $n_\mathrm{out} > 0$, the matrix factorization becomes the bottleneck of both integration schemes. The computational complexity of this schemes is thus

$$\begin{aligned}\mathcal{C}_\mathrm{irk} &= (s(n_\mathrm{x}+n_\mathrm{z}))^3,\\ \mathcal{C}_\mathrm{gnsf} &= (sn_\mathrm{out})^3,\end{aligned}$$

which results in the following complexity ratio

$$\mathcal{C}_\mathrm{ratio} = \frac{\mathcal{C}_\mathrm{gnsf}}{\mathcal{C}_\mathrm{irk}} = \left(\frac{n_\mathrm{out}}{n_\mathrm{x}+n_\mathrm{z}}\right)^3. \tag{3.62}$$

Note that this ratio is only driven by the ratio $\frac{n_\mathrm{out}}{n_\mathrm{x}+n_\mathrm{z}}$, in which it enters cubically. This suggests a large saving in computational cost for systems that can be transcribed into the GNSF form with a ratio $\frac{n_\mathrm{out}}{n_\mathrm{x}+n_\mathrm{z}} \ll 1$, i.e. systems with many states but very few nonlinear dependencies in the dynamics.
However, this only holds asymptotically and for the number of stages s fairly low in typical applications. It does not take sensitivity propagation into account and there are several operations that have to be carried out within GNSF-IRK, for which there is no equivalent in a standard IRK scheme. Although the computational cost of these operations can be neglected asymptotically, it is very relevant for practical applications. This can be observed when comparing the results in Chapter 5 with the theoretical maximum speedup in (3.62).

3.4.3 Lagrange Mechanics and Flexibility of GNSF

In this subsection, we want to recall the remark on mechanical systems in the original NSF paper [47] and extend it to illustrate the flexibility of the GNSF structure. It was already pointed out that mechanical systems are typically derived via Lagrange mechanics, yielding an ODE of the form

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(\dot{q},q,u) = 0, \tag{3.63}$$

where the generalized coordinates $q \in \mathbb{R}^{n_\mathrm{q}}$ are used to define the state vector as $x = \left[q^\top, \dot{q}^\top\right]^\top \in \mathbb{R}^{2n_\mathrm{q}}$.

Such an ODE can be written in the following structured form

$$\begin{bmatrix} \mathbb{1} & \mathbb{0} \\ \mathbb{0} & M(x) \end{bmatrix} \dot{x} = \begin{bmatrix} \mathbb{0} & \mathbb{1} \\ \mathbb{0} & -C(x) \end{bmatrix} x - \begin{bmatrix} \mathbb{0} \\ F(x,u) \end{bmatrix}, \tag{3.64}$$

in which the first n_q equations are linear. Thus, one can transcribe the system into the NSF or GNSF form with ϕ mapping to $\mathbb{R}^{n_\mathrm{q}}$ or potentially even a lower dimensional space. For this kind of systems, the complexity ratio (3.62) that corresponds to the saving in the factorization of the Newton matrix is $\frac{1}{8}$.
Moreover, dynamic models of the form above can be extended by adding more differential states. These could be states that are added after the derivation of the mechanical model in (3.63), as it is the case in the test models presented in Section 5.1.2 and Section 5.1.3, regarding ϑ there.
Additionally, the dynamic model can be extended by adding so-called *quadrature states*. These can be used to track quantities that are not represented by a state of the model but are of interest within the NMPC scheme [48]. Such a quantity could be any kind of objective or constraint function of the general OCP (1.2). This quantity will then be simulated by the integrator, resulting in a more accurate representation compared to the approach of evaluating the corresponding function using the output of the integrator. Since the initial model is not depending on the dynamics of the quadrature states, they can be efficiently simulated using GNSF-IRK by modeling them as part of the LOS.

3.4.4 `GNSF_early` – An Alternative GNSF Formulation

As mentioned in Section 3.2.3, some design choices were adjusted when experimenting with prototypes of GNSF-IRK.
In this subsection, we want to discuss an alternative approach compared to the GNSF-IRK scheme presented in Section 3.3 and its implementation in `acados`. The approach uses `CasADi` code generation more extensively and was used in some early prototype of GNSF-IRK. Thus, we refer to it as `GNSF_early`.

`GNSF_early` – Formulation The idea is to generate the residual function $\mathbf{r}(\mathbf{v}, x_n^{[1]}, u)$ directly in `CasADi` and to omit the linear input matrices. The main argument for this is that `CasADi` can automatically detect the sparsity patterns of matrices and generate efficient `C` code to perform operations using these matrices.
The structured dynamic system for `GNSF_early` uses the same formulation (3.12), but instead of the term $\phi(L_{\dot{\mathrm{x}}}\dot{x}^{[1]} + L_\mathrm{x} x^{[1]} + L_\mathrm{z} z^{[1]}, L_\mathrm{u} u)$, one can use $\phi_{\mathrm{early}}(\dot{x}^{[1]} + x^{[1]} + z^{[1]}, u)$, i.e. no linear input matrices. Since the values for $x^{[1]}, \dot{x}^{[1]}$ are available when evaluating

f_{LO}, it is not beneficial to use linear input matrices for this function.

`GNSF_early` **- Discussion** Note that all points but the last (Linear Input Matrices) from Section 3.2.3 are still valid for the `GNSF_early` formulation.
In every Newton iteration of GNSF-IRK the residual function $\mathbf{r}$ from (3.27) has to be evaluated once. Therefore, the input $\mathbf{y}$ has to be calculated from $\mathbf{v}, x_n^{[1]}, u$, which happens through matrices $\mathbf{Y}_\mathbf{v}, \mathbf{Y}_\text{x}, \mathbf{Y}_\text{u}$. It could make sense to exploit the sparsity of the corresponding matrix multiplications, i.e. in Equation (3.25).
Moreover, the derivatives of the residual function $\frac{\partial \mathbf{r}}{\partial(\mathbf{v}, x_n^{[1]}, u)}$ could be generated using AD in `CasADi` instead of realizing the matrix-matrix multiplication in Equation (3.29) by BLASFEO routines. The code generation could be computationally beneficial compared to the the current `acados` implementation using BLASFEO routines.
We also have to multiply with the matrices $\mathbf{K}_\mathbf{v}, \mathbf{K}_\text{x}, \mathbf{K}_\text{u}, \mathbf{Z}_\mathbf{v}, \mathbf{Z}_\text{x}, \mathbf{Z}_\text{u}$ in order to do both, obtain the output of the scheme via Equations (3.21) and (3.23) and to implement sensitivity propagation, via Equations (3.45) and (3.36).
Additionally, the linear transformations to get the input $\mathbf{y}$ and initial linear operations within ϕ could be summarized by `CasADi`, such that internally the concatenation of these transformations is computed. However, since the `CasADi` developers highlight that `CasADi` is not a computer algebra system (CAS) [28], it is not expected by the author that this happens within the code generation, but it could be further investigated.
In Section 5.6, we will observe that all the bold matrices are often quite sparse, especially for large-scale models. Thus, there is some potential for sparsity exploitation. However, it will strongly depend on the dynamic model, if code generated `CasADi` functions would outperform the high-performing dense linear algebra BLASFEO routines or vice versa.
The final implementation of GNSF-IRK follows the paradigms maintainability and adaptivity of `acados` more closely instead of code generation, which was the paradigm of ACADO and this early version.

Comparison to GNSF–IRK and its `acados` implementation In the remainder of this subsection, we want to give some reasons for using the formulation with linear input matrices in (3.12) throughout this thesis. In the following, we assume that the code generation should happen before choosing the integrator options, i.e. Butcher tableau, step size T_{int}, etc.
First, since we have to perform a precomputed linear transformation on $x_0, \mathbf{v}, u$ anyway for each evaluation of the gathered nonlinearity function Φ, see (3.26), we can include linear transformations into the precomputed one with no additional

computational cost. The linear transformation included through the linear input matrices should be one that would have to be performed within the nonlinearity function otherwise.
Second, this formulation enables us to reduce the input dimension of ϕ, using only the components which are needed. The input dimension n_{in} has $2n_{\text{x}_1} + n_{\text{u}} + n_{\text{z}_1}$ as an upper bound when reformulating the model reasonably, i.e. taking Section 4.1 into account. Note that $2n_{\text{x}_1} + n_{\text{u}} + n_{\text{z}_1}$ is the input dimension of ϕ using the other formulation `GNSF_early`. This makes the matrices corresponding to the linear transformations from our previous point likely to be smaller and thus the matrix multiplications cheaper. Using the same argument, the Jacobian matrices of ϕ get smaller and the corresponding matrix-matrix multiplications, see (3.29), become cheaper.
This being said, using linear input matrices is beneficial, assuming that we want to use code generation only for the model functions ϕ, f_{LO}.
In practice, it is very uncomfortable to switch from `C` to MATLAB and back to `C` when changing options of the integrator, which is one reason why there is no extensive comparison between the two versions. Furthermore, modularity, maintainability and adaptivity are key concepts of `acados` and using the `GNSF_early` approach would clash with these concepts.
Regarding the recent developments in `acados`, one way to combine the `GNSF_early` approach with the `acados` framework reasonably would be to perform the automatic transcription and code generation using `CasADi` in the `C++` interface layer. This layer is still under development and its main purpose is to generate the MATLAB and `Python` interfaces through `SWIG` [49].

3.4.5 Further possible Generalizations and Improvements

Compatibility with Simplified and Single Newton methods In order to exploit the structure of the IRK equations even more, one could try to combine the two approaches presented in Section 3.1. Note that the three-stage dynamic system structure exploitation presented in Section 3.1.2 and [46] can be combined with the IRK structure exploitation presented in Section 3.1.1 using the DAE formulation in [38].
However, it is not clear if and how the scheme in [47] and the one presented in this thesis can be efficiently combined with the IRK structure exploitation in Section 3.1.1, which could be further investigated.

A Multi-Stage GNSF Structured System It is possible to generalize the GNSF structure (3.12) further by concatenating an arbitrary number of GNSF systems, as

it is shown in Figure 3. This idea has been suggested in [12, p.134] for the three-stage dynamic system structure presented in Section 3.1.2.
The implementation of a multi-stage GNSF-IRK scheme is a task that remains outside the scope of this thesis. Moreover, it is even harder to transcribe a dynamic model into the multi-stage GNSF structure shown in Figure 3, i.e. the algorithm in Section 4.2 has to be thoughtfully extended. Generally speaking, it is not clear if such structures appear frequently within dynamic models used in practice. However, we expect that this structure is often present within multibody models that are used often in robotics [50].
Thus, we suggest for future work to first extend the automatic transcription method from Section 4.1 to multi-stage GNSF structured systems and investigate how far and often such structures can be detected within models of interest.

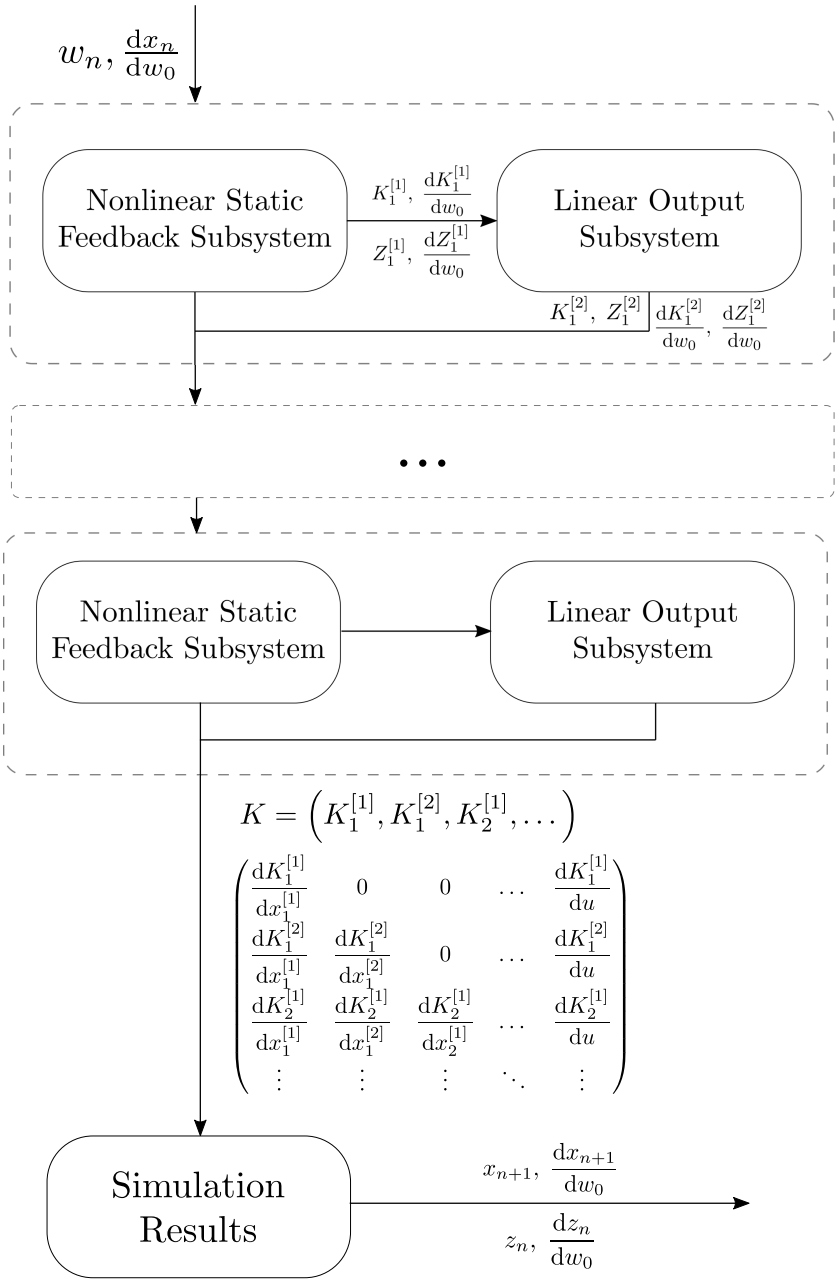

Figure 3: Concept of a multi-stage GNSF structured dynamic system and workflow of an IRK scheme exploiting this structures.

4 GNSF Model Transcription

In this chapter, we first discuss general principles one should take into account when transcribing a model into the GNSF structure and then derive an automatic transcription method that has been implemented as a feature of `acados`.

4.1 General Transcription Discussion

For an efficient deployment of GNSF-IRK the given dynamic model has to be transcribed into the GNSF structure (3.12). 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 speed-up by using the GNSF-IRK scheme:

1. **Maximize components modeled in LOS** (3.12b)**:** 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 (3.12a) or of the LOS (3.12b). As 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 dimension of Newton iteration:** The output dimension n_{out} of the nonlinearity function ϕ should be as small as possible, as the computational complexity is cubic in n_{out}.

3. **Minimize input dimension of ϕ:** The input dimensions $n_{\text{y}}, n_{\hat{\text{u}}}$ of ϕ should be as small as possible in order to minimize the size of the linear input matrices $L_{\dot{\text{x}}}, L_{\text{x}}, L_{\text{z}}, L_{\text{u}}$ and the cost of multiplying with $\mathbf{Y_v}, \mathbf{Y}_{\text{u}}, \mathbf{Y}_{\text{x}}$ (introduced earlier in this section) and L_{u}, which has to be done in each Newton iteration and within the sensitivity propagation.

4. **Outsource initial linear transformations into the linear input matrices:** This can help to tune the reformulation with respect to the point above. Sometimes, the nonlinearity function ϕ actually proceeds a weighted sum of components and the input dimension of ϕ can be reduced by taking this into account when defining the linear input matrices.

4.2 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, the goal is to derive an algorithm that transcribes such models into the GNSF structure (3.12) and follows the principles derived in the previous section.
This section introduces the concepts of an algorithm, that follows the principles 1 to 3 from the previous section and is referred to as *automatic transcription* method. Additionally, we present its implementation as pseudo code structured into functions in the Algorithms 4 to 10. Algorithm 4 gives an overview of the transcription algorithm using the functions presented in Algorithms 5 to 10.

Algorithm 4: Transcribe a dynamic system into the GNSF structure

1: **Input:** Implicit model consisting of $x, \dot{x}, u, z$ and an expression $f^{\text{impl}}(x, \dot{x}, u, z)$ describing the dynamics.
2: `gnsf ← determine_trivial_gnsf_transcription( model )` ▷ Alg. 5
3: `gnsf ← detect_affine_terms_reduce_nonlinearity( gnsf )` ▷ Alg. 7
4: `[gnsf, reordered_model] ← reformulate_with_LOS( gnsf, model )` ▷ Alg. 8,9
5: `gnsf ← reformulate_with_invertible_E_mat( gnsf )` ▷ Alg. 10
6: `check_reformulation( gnsf, reordered_model )`
7: `generate_C_code( gnsf, reordered_model )`
8: **Output:**

- reordered_model: consisting of a sorted state vector $\bar{x}$ and a permuted expression $\bar{f}_{\text{impl}}$ in which some functions which were made part of the linear output system of the GNSF model have changed signs
- gnsf: dynamic system model in GNSF format, equivalent to the reordered model

Pseudo Code Notation In the following code, the concept of index sets is realized by using lists or vectors. For notational convenience, the basic set operations $\setminus$, $\cup$, $\cap$, $\in$ are naturally extended for these types of objects . Additionally, $-$ is used to denote the subtraction of a scalar from each element of the list.

Initial Transcription In the first function (Algorithm 5), the dynamic system is trivially transcribed into the GNSF structure, i.e. with $A, B, C, E, c = \mathbb{0}$, $C = \mathbb{1}$ and an empty LOS.

Algorithm 5: Function `determine_trivial_gnsf_transcription( model )`

1: **Input:** model: Implicit dynamic system model consisting of $x, \dot{x}, u, z$ and an expression $f^{\text{impl}}(x, \dot{x}, u, z)$ describing the dynamics.

`Obtain trivial transcription`

2: $x^{[1]} \leftarrow x,\ z^{[1]} \leftarrow z,\ x^{[2]} \leftarrow [\ \ ],\ z^{[2]} \leftarrow [\ \ ];$ ▷ empty LOS

3: $A \leftarrow \mathbb{0},\ B \leftarrow \mathbb{0},\ C \leftarrow \mathbb{1},\ E \leftarrow \mathbb{0},\ c \leftarrow \mathbb{0},\ \phi \leftarrow f^{\text{impl}};$ ▷ NSF (3.12a)

4: $E^{\text{LO}} \leftarrow [\ \ ],\ A^{\text{LO}} \leftarrow [\ \ ],\ f_{\text{LO}} \leftarrow [\ \ ];$ ▷ LOS (3.12b)

5: create "gnsf" containing struct $E, A, B, C, c, E^{\text{LO}}, A^{\text{LO}}, \phi, f_{\text{LO}}, x, \dot{x}, u, z$ ▷ Define gnsf model

6: gnsf ← `determine_input_nonlinearity_function`(gnsf) ▷ Alg. 6

7: `check_reformulation`(gnsf, reordered_model)

8: **Output:** gnsf: dynamic system model in GNSF format, equivalent to the model

Afterwards (in Algorithm 7), the transcription method detects all affine terms in the nonlinearity function ϕ and represents them through the matrices A, B, E, c instead. Subsequently, all structural zeros in the expression ϕ is detected and removed, reducing the dimension n_{out}. The input of the nonlinearity function is determined in Algorithm 6. Here, y ($\hat{u}$) is defined as the concatenation of $x, \dot{x}, z$ (u) components, that the expression ϕ depends on. The linear input matrices $L_{\text{x}}, L_{\dot{\text{x}}}, L_{\text{z}}, L_{\text{u}}$ are selecting the elements of $x, \dot{x}, z$ and u respectively. Thus, they are binary matrices and the idea to outsource initial linear transformations into these matrices is not taken into account by the proposed algorithm.

Algorithm 6: Function `determine_input_nonlinearity_function`

Input: gnsf dynamic system model without input of nonlinearity $y, \hat{u}$ and corresponding matrices $L_{\mathrm{x}}, L_{\dot{\mathrm{x}}}, L_{\mathrm{z}}, L_{\mathrm{u}}$

$y \leftarrow [\ ], \quad \hat{u} \leftarrow [\ ]$ ▷ initialize empty

`Determine` y

for $v \in \{x^{[1]}, \dot{x}^{[1]}, z\}$ **do**
  for $i = 1 : \mathrm{length}(v)$ **do** ▷ check dependency on v_i
    if $\frac{\partial \phi}{\partial v_i} \neq \mathbb{0}$ **then**
      $y \leftarrow \left[y,\ v_i^{[1]}\right]$ ▷ append v_i to y
    end if
  end for
end for

`Determine` $\hat{u}$

for $i = 1 : \mathrm{length}(u)$ **do** ▷ check dependency on u_i
  if $\frac{\partial \phi}{\partial u_i} \neq \mathbb{0}$ **then**
    $\hat{u} \leftarrow \left[\hat{u},\ u_i^{[1]}\right]$ ▷ append u_i to $\hat{u}$
  end if
end for

`Determine linear input matrices`

$L_{\mathrm{x}} \leftarrow \frac{\partial y}{\partial x^{[1]}}, \quad L_{\dot{\mathrm{x}}} \leftarrow \frac{\partial y}{\partial \dot{x}^{[1]}}, \quad L_{\mathrm{z}} \leftarrow \frac{\partial y}{\partial z}, \quad L_{\mathrm{u}} \leftarrow \frac{\partial \hat{u}}{\partial u}$

make $y, \hat{u}, L_{\mathrm{x}}, L_{\dot{\mathrm{x}}}, L_{\mathrm{z}}, L_{\mathrm{u}}$ part of gnsf

Output: gnsf dynamic system model with input of nonlinearity $y, \hat{u}$ and corresponding matrices $L_{\mathrm{x}}, L_{\dot{\mathrm{x}}}, L_{\mathrm{z}}, L_{\mathrm{u}}$

Algorithm 7: Function `detect_affine_terms_reduce_nonlinearity( gnsf )`

Input: gnsf: dynamic system model in GNSF format, with trivial model matrices $A, B, C, c, E, A^{\mathrm{LO}}$

`Detect linear terms in` x

for $i = 1 : \mathrm{length}(\phi)$ **do** ▷ determine A
  for $j = 1 : \mathrm{length}(x)$ **do**
    if $\frac{\partial \phi_i}{\partial x_j}$ is constant **then**
      $A(i,j) \leftarrow \frac{\partial \phi_i}{\partial x_j}$;
    end if
  end for
end for

`Detect linear terms in` $u, \dot{x}, z$

Similar to the above, details omitted here ▷ determine B, E

`Detect constant terms`

for $i = 1 : \mathrm{length}(\phi)$ **do** ▷ determine c
  if ϕ_i is constant **then**
    $c_i \leftarrow \phi_i$;
  end if
end for

$\phi \leftarrow \phi - Ax - Bu + E \begin{bmatrix} \dot{x} \\ z \end{bmatrix} - c$; ▷ define consistent ϕ

`Reduce dimension of nonlinearity`

$I_{=0} \leftarrow \{ i \mid \phi_i = 0 \}, \quad I_{\neq 0} \leftarrow \mathbb{Z}_1^{n_{\mathrm{out}}} \setminus I_{=0}$; ▷ detect zeros

$\phi \leftarrow \phi_{I_{\neq 0}}, \quad n_{\mathrm{out}} \leftarrow \mathrm{length}(\phi)$, ▷ reduce nonlinearity
$C \leftarrow \mathbb{0}_{(n_{\mathrm{x}_1}+n_{\mathrm{z}}) \times n_{\mathrm{out}}}$;

for $i = 1 : n_{\mathrm{out}}$ **do** ▷ define consistent C
  $C(I_{\neq 0}(i), i) \leftarrow 1$
end for

`Update gnsf model`

make A, B, C, c, E, ϕ part of gnsf

`gnsf ← determine_input_nonlinearity_function( gnsf )`

`check_reformulation(gnsf, reordered_model)`

Output: gnsf: dynamic system model in GNSF format, where all affine linear dependencies are moved from ϕ to the matrices A, B, c, E

Detection of the Linear Output System Now that the linear structure is detected, a function, presented in Algorithms 8 and 9, needs to be called that detects a set of x, z components that can be made part of the LOS.
In this function, we first check the necessary criterion for components to be made part of the LOS. This is that the nonlinear functions ϕ, f_{LO} can not depend on any of these components. This criterion is used to determine components that in any case have to be part of the NSF subsystem, $I_{\mathrm{NSF\_comp}}$, and components that can potentially be modeled as through the LOS I_{cand} (first part of Algorithm 8).
When reformulating the structured dynamic system with a LOS, additionally to determining the division of the states, the equations have to be divided accordingly. Here, the approach is to determine one equation for each component that will be kept as part of the NSF type system. It is necessary to determine the equations, as they have to be checked for dependencies on the candidates $i \in I_{\mathrm{cand}}$ later. If one of the NSF equations ($I_{\mathrm{NSF\_eq}}$) depends on one of the candidates, this candidate has to be modeled through the NSF type subsystem, see Algorithm 8, line 26.
In the easiest case, there is exactly one equation in which $\dot{x}_i$ occurs linearly, then this equation will be the one associated with $\dot{x}_i$, see Algorithm 8, line 9.
The second case is that there is more than one equation with a linear term in $\dot{x}_i$, (respectively z_i). In this case, we want to choose the equation that remains part of the NSF type system to be the one that has the least dependencies on candidates for the LOS. Therefore, we define the following quantity, indicating how dependent the equation with index i_{eq} is on the candidates for the LOS I_{cand}

$$\zeta(i_{\mathrm{eq}}, I_{\mathrm{cand}}) := |\{\, i \in I_{\mathrm{cand}} \mid \exists \text{ linear term } x_i \text{ or } \dot{x}_i, \text{ respectively } z_i \text{ in equation } i_{\mathrm{eq}} \,\}| \tag{4.1}$$

which one could call *candidate dependency* of equation i_{eq} given the candidates I_{cand}. Equations with lower candidate dependency index are preferred to be kept part of the NSF type system, see Algorithm 8 line 12.
The third case is that there is no equation with a linear term in $\dot{x}_i$, respectively z_i. In this case, we check in which of the unsorted equations a nonlinear dependency on $\dot{x}_i$, respectively z_i, occurs, see Algorithm 8, line 14. Given this set of equations, the one with the least candidate dependency is taken. Subsequently, the equation is modified by adding the term $\dot{x}_i$, respectively z_i, to both sides of the equation, i.e. via E and $C\phi$, see Algorithm 8, line 20. This is done, because the equation i_{eq} corresponding to a NSF component $\dot{x}_i$, respectively z_i, should have a nonzero entry in the corresponding entry of $E_{i_{\mathrm{eq}},:}$. Otherwise, it is likely to end up with a singular

matrix E. For example, when attempting to model an algebraic equation of the form

$$0 = g(z_i, \bar{x}),$$

where $\bar{x}$ is a subvector of the state vector x and g is nonlinear in both $\bar{x}$ and z_i. Transcribing this equation without the mentioned trick would result in a zero row in the matrix E.
Given the NSF components and an equal number of associated equations, the remaining equations can be rewritten as a linear output type system (3.12b). Here, all terms can be transcribed using the nonlinear function f_{LO}, except the linear terms in $\dot{x}^{[2]}, z^{[2]}$ and $x^{[2]}$, which are modeled using E^{LO} and A^{LO} respectively. The detailed transcription can be found in Algorithm 9, lines 35 to 38.

Ensuring regularity of $E_{11}, E_{22}, E^{\text{LO}}$ To ensure that the necessary conditions on the GNSF model formulation hold, i.e. that $E_{11}, E_{22}, E^{\text{LO}}$ are regular, Algorithm 10 was added to the automatic GNSF transcription algorithm.
In the first part, E_{11} and E_{22} are checked for regularity. Let us regard one of them and refer to it as M. If M is not regular, the algorithm will find the submatrix $M_{\text{sub}} = M_{\mathbb{Z}_1^j, \mathbb{Z}_1^j}$ with minimal j, such that M_{sub} is not regular. Then, it will add a 1 to the lower right entry of M_{sub} and a corresponding term to the $C\phi$ part of the corresponding equation in the GNSF model.

Remark 4.1 Algorithm 8 ensures that an equation i which is part of the NSF part corresponds to the variable v_i, due to the sorting in lines 23 and 32. Additionally, each of these equations has a nonzero entry in the column of E corresponding to v_i, i.e. $E_{i,i} \neq 0$. Thus, all diagonal entries of E are nonzero. This makes it likely that E_{11} and E_{22} are regular before the function in Algorithm 10 is called. In this case Algorithm 10 just checks the regularity of the matrices E_{11}, E_{22} and E^{LO} once by evaluating their determinant.

Remark 4.2 Algorithm 10 is based on a heuristic and the modified matrix E_{11} or E_{22} might be ill-conditioned. Thus, if the function actually modifies the model, it will always print a warning, reminding the user that it should be used carefully and one ought to consider checking the model formulation.

Additionally, the matrix E^{LO} will be checked for regularity. In case that this check fails, the algorithm will throw an error because it is not possible to add terms in $\dot{x}_i^{[2]}$ to both sides, since there is no term the right side of (3.12b) that can handle expressions in $\dot{x}^{[2]}$.

Theorem 4.3 *Given an index-1 DAE, Algorithm 8 ensures that there are no zero columns in* E^{LO}.

Proof. Assuming a component v_i of $\dot{x}$ or z is made part of the LOS. Then this component did not occur nonlinearly in any of the model equations. Otherwise, there is a contradiction to line 2 of Algorithm 8. Additionally, there can be no linear dependency on v_i in (3.12a), this would contradict line 26 of Algorithm 8. Thus, if there is a zero column in E^{LO}, the corresponding variable did not occur at all in the initial model, which contradicts the assumption that the model was an index-1 DAE. □

Corollary 4.4 *The matrices* $E_{11}, E_{22}, E^{\mathrm{LO}}$ *that are generated by the proposed automatic transcription method have no zero column.*

Implementation and Limitations The transcription algorithm is implemented as a MATLAB function for `CasADi` models in `SX` symbolic variables. It was made part of `acados` and can be found in the folder `/interfaces/matlab/sim` in the `acados` Github repository [2].
When testing the transcription algorithm, we found the following limitations:

- **Limitation to `SX` models:** It is not possible to use the algorithm with `MX` symbolic variables. For these expressions the `CasADi` function `simplify` is less powerful compared to the implementation for `SX` expressions. For example, after subtracting some summands, `CasADi` is not able to detect structural zeros in the expression corresponding to the nonlinearity function ϕ. This is due to the fact, that `CasADi`'s `MX` symbolic variables use matrix operations by default, for which the CAS functionality is quite limited.
  Within most wind turbine models splines are used to evaluate the aerodynamic coefficients, see Section 5.1. Unfortunately, splines and external functions in `CasADi` are only supported with `MX` symbolic variables.
  One solution to this problem is to define the model with `SX` symbolic variables and a dummy spline function such that the algorithm can be used. Subsequently one would have to parse the resulting model and define an equivalent `MX` model. However, a better solution would be to implement an interface for the spline such that it can be used with `SX` symbolic variables.

- **Limitation to models with explicit coefficients:** Another limitation of this implementation is also connected to a flaw of the `CasADi` function `simplify`.

For example, given an equation of the form

$$\frac{(c_1 x_1 + c_2 x_2)}{c_3} = 0, \tag{4.2}$$

whereby x_1, x_2 are components of the state vector x and $c_1, c_2, c_3 \in \mathbb{R}$ are constants, Algorithm 7 will find that the coefficient associated with x_1 is equal to $\frac{c_1}{c_3}$. However, the `CasADi` function `simplify` cannot simplify the expression $\frac{(c_1 x_1 + c_2 x_2)}{c_3} - \frac{c_1 x_1}{c_3}$ to $\frac{c_2 x_2}{c_3}$. Thus, in the initial `CasADi` expression f^{impl}, the coefficients should be multiplied out.

The underlying problem is probably connected to the fact that a linear term in a variable can be represented through multiple nodes of the expression graph, which is `CasADi`'s internal representation of an expression.

One solution to this deficit of the `CasADi` CAS capabilities would be to use the `CasADi` expression, for which Algorithm 7 detected a linear dependency on x_i. This expression could be parsed and the summand that contains x_i could be removed such that an equivalent `CasADi` expression without the term is obtained.

However, the most elegant solution would probably be to empower `CasADi`'s CAS to simplify such terms.

A solution that tackles both of the above limitations is to implement the transcription algorithm using a different full featured CAS that is able to simplify such terms. It should however be possible to interface this CAS with `CasADi` such that the matrices and expressions can be automatically transfered into `CasADi`, which in turn will generate the efficient `C` code for `acados`.

As an example, the application of the proposed automatic transcription method to a test problem is illustrated in Section 5.1.1.

Algorithm 8: Part 1 of Function `reformulate_with_LOS( gnsf, model )`

Input: gnsf: dynamic system model in GNSF format, where all affine linear dependencies are moved from ϕ to the matrices A, B, c, E and the LOS is empty

`Check necessary criterion`

$I_{\text{NSF\_comp}} \leftarrow \{\, i \mid \phi \text{ depends on } x_i \text{ or } \dot{x}_i \,\}$;
$I_{\text{NSF\_comp}} \leftarrow I_{\text{NSF\_comp}} \cup \{\, i + n_{\text{x}} \mid \phi \text{ depends on } z_i \,\}$
$I_{\text{cand}} \leftarrow \mathbb{Z}_1^{n_{\text{x}}+n_{\text{z}}} \setminus I_{\text{NSF\_comp}}$;
$I_{\text{NSF\_eq}} \leftarrow \varnothing, \quad {I_{\text{NSF\_comp}}}^{+} \leftarrow I_{\text{NSF\_comp}}, \quad I_{\text{unsort\_eq}} \leftarrow \mathbb{Z}_1^{n_{\text{x}}+n_{\text{z}}}$;
while true **do**
  for $i \in {I_{\text{NSF\_comp}}}^{+}$ **do**

`Find equation corresponding to component` i

    $I_{\text{eq}} \leftarrow \{\, j \mid E_{j,i} \neq 0 \,\} \cap I_{\text{unsort\_eq}}$;
    if $|I_{\text{eq}}| == 1$ **then**
      $i_{\text{eq}} \leftarrow I_{\text{eq}}$
    else if $|I_{\text{eq}}| > 1$ **then**
      Determine $\zeta(j, I_{\text{cand}})$ for $j \in I_{\text{eq}}$
      $i_{\text{eq}} \leftarrow \arg\min_{j \in I_{\text{eq}}} \zeta(j, I_{\text{cand}})$
    else
      $I_{\text{eq}} \leftarrow \{\, j \in I_{\text{unsort\_eq}} \mid \text{component } i \text{ occurs nonlinearly in eq. } j \,\}$
      if $I_{\text{eq}} = \varnothing$ **then**
        $I_{\text{eq}} \leftarrow I_{\text{unsort\_eq}}$
      end if
      Determine $\zeta(j, I_{\text{cand}})$ for $j \in I_{\text{unsort\_eq}}$
      $i_{\text{eq}} \leftarrow \arg\min_{j \in I_{\text{eq}}} \zeta(j, I_{\text{cand}})$
      Add term $\dot{x}_i$ (respectively z_i) to both sides of i_{eq} (via E and $C\phi$)
    end if
    $I_{\text{NSF\_eq}} \leftarrow I_{\text{NSF\_eq}} \cup \{i_{\text{eq}}\}, \quad I_{\text{unsort\_eq}} \leftarrow I_{\text{unsort\_eq}} \setminus \{i_{\text{eq}}\}$
    Store the pair (i, i_{eq});
  end for

`Add NSF components`

  for $i_{\text{eq}} \in I_{\text{NSF\_eq}}$ **do**
    $I_{\text{NSF\_comp}} \leftarrow I_{\text{NSF\_comp}} \cup \{\, j \mid E_{i_{\text{eq}},j} \neq 0 \text{ or } A_{i_{\text{eq}},j} \neq 0 \,\}$;
    ${I_{\text{NSF\_comp}}}^{+} \leftarrow I_{\text{cand}} \cap I_{\text{NSF\_comp}}$;
  end for
  if ${I_{\text{NSF\_comp}}}^{+} == \varnothing$ **then** break;
  end if
end while
Sort the list $I_{\text{NSF\_eq}}$ according to the indices $I_{\text{NSF\_comp}}$ using line 23 ;

Algorithm 9: Part 2 of Function `reformulate_with_LOS( gnsf, model )`

`Final split`

$I_{\text{LOS\_comp}} \leftarrow I_{\text{cand}}$, $I_{\text{LOS\_eq}} \leftarrow \mathbb{Z}_1^{n_\text{x}+n_\text{z}} \setminus I_{\text{NSF\_eq}}$
$I_{\text{x1}} \leftarrow I_{\text{NSF\_comp}} \cap \mathbb{Z}_1^{n_\text{x}}$, $I_{\text{z1}} \leftarrow I_{\text{NSF\_comp}} \cap \mathbb{Z}_{1+n_\text{x}}^{n_\text{x}+n_\text{z}} - n_\text{x}$
$I_{\text{x2}} \leftarrow I_{\text{LOS\_comp}} \cap \mathbb{Z}_1^{n_\text{x}}$, $I_{\text{z2}} \leftarrow I_{\text{LOS\_comp}} \cap \mathbb{Z}_{1+n_\text{x}}^{n_\text{x}+n_\text{z}} - n_\text{x}$

`Permute states` $x, \dot{x}, z$

$x \leftarrow [x_{I_{\text{x1}}}; x_{I_{\text{x2}}}]$, $\dot{x} \leftarrow [\dot{x}_{I_{\text{x1}}}; \dot{x}_{I_{\text{x2}}}]$, $z \leftarrow [z_{I_{\text{z1}}}; x_{I_{\text{z2}}}]$

`Redefine equations` $I_{\text{LOS\_eq}}$ `in LOS`

for $i_\text{eq} \in I_{\text{LOS\_eq}}$ **do**
  $i_\text{lo} \leftarrow$ index of i_eq in $I_{\text{LOS\_eq}}$
  $f_\text{LO} \leftarrow \left[f_\text{LO};\ (A_{i_\text{eq},I_\text{x1}} x^{[1]} + B_{i_\text{eq},:} u + c_{i\text{eq}} + C_{i_\text{eq},:}\phi - E_{i_\text{eq},I_\text{x1}} \dot{x}^{[1]} - E_{i_\text{eq},I_\text{z1}} z^{[1]})\right]$
  $A^\text{LO}_{i_\text{lo},:} \leftarrow A_{i_\text{eq},I_\text{x2}}, \quad E^\text{LO}_{i_\text{lo},:} \leftarrow E_{i_\text{eq},I_{\text{LOS\_comp}}}$
end for

`Define reordered model`

$f^\text{impl} \leftarrow \begin{bmatrix} f^\text{impl}{}_{I_{\text{NSF\_eq}}}; & f^\text{impl}{}_{I_{\text{LOS\_eq}}} \end{bmatrix}$

Set up reordered_model consisting of $x, \dot{x}, z, u, f^\text{impl}$

`Remove LOS equations from NSF`

Remove rows corresponding to $I_{\text{LOS\_eq}}$ from A, B, C, E, c

`Reduce nonlinearity` $C\phi$

$I_{\neq 0} \leftarrow \{\, i \mid \phi_i \neq 0 \,\}$, $C \leftarrow C_{I_{\neq 0}}$, $\phi \leftarrow \phi_{I_{\neq 0}}$

`Update gnsf model`

make new $E, A, B, C, c, \phi, E^\text{LO}, A^\text{LO}, f, x, \dot{x}$ part of gnsf
`gnsf ← determine_input_nonlinearity_function( gnsf )`

Output:

- reordered_model: consisting of a sorted state vector $\bar{x}$, a permuted expression $\bar{f}_\text{impl}$
- gnsf: dynamic system model in GNSF format, equivalent to the reordered model, where as many states as possible are made part of LOS $x^{[2]}$

Algorithm 10: Function `reformulate_with_invertible_E_mat( gnsf )`

Input: gnsf: dynamic system model in GNSF format, with possibly non-invertible matrices E, E^{LO}.
$k \leftarrow \begin{bmatrix} \dot{x}^{[1]} \\ z \end{bmatrix}$
for $i \leftarrow 1, 2$ **do**
  if $i == 1$ **then**
    $I \leftarrow \mathbb{Z}_1^{n_{\mathrm{x}_1}}$
  else if $i == 2$ **then**
    $I \leftarrow \mathbb{Z}_{n_{\mathrm{x}_1}+1}^{n_{\mathrm{x}_1}+n_{\mathrm{z}}}$
  end if
  if $\det(E_{I,I}) == 0$ **then**
    for $i_{\max} \in I$ **do**
      $I_{\mathrm{sub}} \leftarrow \mathbb{Z}_{\min(I)}^{i_{\max}}, \quad E_{\mathrm{sub}} \leftarrow E_{I_{\mathrm{sub}},I_{\mathrm{sub}}}$
      while $\mathrm{rank}(E_{\mathrm{sub}}) < \mathtt{length}(I_{\mathrm{sub}})$ **do**
        $E_{i_{\max},i_{\max}} \leftarrow E_{i_{\max},i_{\max}} + 1$
        if $C_{i_{\max},:} == \mathbb{0}$ **then** ▷ Add new entry to ϕ
          $\phi \leftarrow [\phi;\ k(i_{\max})]$
          $C_{:,\mathtt{length}(\phi)+1} \leftarrow \mathbb{0}, \quad C_{i_{\max},\mathtt{length}(\phi)+1} \leftarrow 1$
        else ▷ Modify entry of ϕ
          $j \leftarrow$ index where $E_{:,j} \neq 0$
          if $C_{:,j} \neq 0$ **then**
            error: C is not a selection matrix!
          end if
          $\phi_j \leftarrow \phi_j + \frac{k_{i\max}}{C_{i_{\max},j}}$
        end if
      end while
    end for

`Update gnsf model`
    Make ϕ, C part of gnsf
    gnsf $\leftarrow$ `determine_input_nonlinearity_function`(gnsf)
  end if
end for
if $\det(E^{\mathrm{LO}}) == 0$ **then**
  Error: Either (probably) a component of $\dot{x}, z$ does not occur in the model at all, **or**
  the columns of E^{LO} happen to be linearly dependent, with no zero column.
end if
Output: gnsf: dynamic system model in GNSF format, with invertible matrix E, or error.

5 Evaluation and Numerical Experiments

In this chapter, the proposed algorithms, GNSF-IRK and the automatic transcription method, are tested on different dynamic system models that are presented in Section 5.1.
All numerical experiments presented in this chapter were performed using `acados` on an ordinary Laptop equipped with an Intel i7-3520M processor, running a 64-bit version of Ubuntu 16.04.
The `C` code to perform the numerical experiments with the public models as well as the implementations of the proposed Algorithms can be found can be found in the Github branch [51]. The GNSF-IRK and standard IRK implementation can be found in `/acados/sim/sim_gnsf.c` and `/acados/sim/sim_irk_integrator.c`, respectively. The files to run the numerical experiments are located in the directory `/examples/c/`.
The Butcher tableaux used correspond to the Gauss-Legendre collocation methods, presented in Section 2.2.5. Within the IRK schemes, the first order forward sensitivity propagation is also performed and included in the computation times.

5.1 Test Problems and Dynamic Models

This section briefly presents the dynamic models that are used to obtain the numerical results presented in the rest of this chapter.

5.1.1 An Inverted Pendulum DAE Model

We refer to the following DAE system as `inv_pend`, as it models a planar pendulum. The model formulation consists of six differential states

$$\begin{aligned} (p_{\mathrm{x}}, v_{\mathrm{x}}),\ (p_{\mathrm{y}}, v_{\mathrm{y}}) &: \text{position and velocity in x- and y-direction,} \\ (\alpha, v_\alpha) &: \text{angular position and velocity,} \end{aligned} \tag{5.1}$$

five algebraic states

$$\begin{aligned} (a_{\mathrm{x}}, a_{\mathrm{y}}) &: \text{acceleration in x- and y-direction,} \\ (a_\alpha) &: \text{angular acceleration,} \\ (F_{\mathrm{x}}, F_{\mathrm{y}}) &: \text{resulting forces in x- and y-direction} \end{aligned} \tag{5.2}$$

and one control input u which denotes the force applied in x-direction from outside. The parameters used in the formulation are the mass $m = 2$, the applied torque $M = 3.5$, the moment of inertia $I = 0.1$ and the gravitational constant $g = 9.81$. The model equations are naturally written in the semi-explicit form as

$$\dot{p}_{\mathrm{x}} = v_{\mathrm{x}}, \tag{5.3a}$$
$$\dot{p}_{\mathrm{y}} = v_{\mathrm{y}}, \tag{5.3b}$$
$$\dot{\alpha} = v_{\alpha}, \tag{5.3c}$$
$$\dot{v}_{\mathrm{x}} = a_{\mathrm{x}}, \tag{5.3d}$$
$$\dot{v}_{\mathrm{y}} = a_{\mathrm{y}}, \tag{5.3e}$$
$$\dot{v}_{\alpha} = a_{\alpha}, \tag{5.3f}$$
$$0 = m a_{\mathrm{x}} - (F_{\mathrm{x}} + u), \tag{5.3g}$$
$$0 = m a_{\mathrm{y}} + m g - F_{\mathrm{y}}, \tag{5.3h}$$
$$0 = I a_{\alpha} - M - (F_{\mathrm{x}} + u) p_{\mathrm{y}} + F_{\mathrm{y}} p_{\mathrm{x}}, \tag{5.3i}$$
$$0 = a_{\mathrm{x}} + v_{\mathrm{y}} v_{\alpha} + p_{\mathrm{y}} a_{\alpha}, \tag{5.3j}$$
$$0 = a_{\mathrm{y}} - v_{\mathrm{x}} v_{\alpha} - p_{\mathrm{x}} a_{\alpha} \tag{5.3k}$$

As an example, the GNSF structured model and the application of the automatic transcription method shall be illustrated for this model in the following.
We used the automatic transcription method presented in Section 4.2 to obtain an equivalent GNSF structured model. The dimensions of the implicit model and the equivalent GNSF model can be found in Table 4.

Table 4: Dimensions of the inverted pendulum (`inv_pend`) model.

Model dimensions				GNSF dimensions				
n_{x}	n_{u}	n_{z}	n_{param}	n_{x_1}	n_{z_1}	n_{out}	n_{y}	$n_{\hat{\mathrm{u}}}$
6	1	5	0	5	5	3	8	1

Regarding the dynamic model in (5.3), we observe that the last three equations contain nonlinear terms, namely products of x and z components. Thus, these equations have to be modeled through the NSF part of GNSF. Since the products of state components are different within all three equations, we can conclude that in an equivalent GNSF structured model $n_{\mathrm{out}} \geq 3$ has to hold. Table 4 states that for the automatically obtained transcription $n_{\mathrm{out}} = 3$, which means that principle 2 from Section 4.1 is fulfilled in this example.
Furthermore, the differential and algebraic states that are involved in these equations also have to be modeled through the NSF part. Sorted as in (5.1) and (5.2), we

conclude that the x components $p_{\mathrm{x}}, p_{\mathrm{y}}, v_{\mathrm{x}}, v_{\mathrm{y}}, v_\alpha$ and z components $a_{\mathrm{x}}, a_{\mathrm{y}}, a_\alpha, F_{\mathrm{x}}, F_{\mathrm{y}}$ have to be part of the NSF part of GNSF.
We reasoned that all states except α have to be part of the NSF part of GNSF. Since the dynamics of α in (5.3c) only depend on v_α and no other equation is depending on α or $\dot{\alpha}$, we observe that it is possible to model α through the LOS part of GNSF. The order of the differential and algebraic states is not changed by the transcription algorithm, i.e.

$$x = \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ v_{\mathrm{x}} \\ v_{\mathrm{y}} \\ v_\alpha \\ \alpha \end{bmatrix}, \quad z = \begin{bmatrix} a_{\mathrm{x}} \\ a_{\mathrm{y}} \\ a_\alpha \\ F_{\mathrm{x}} \\ F_{\mathrm{y}} \end{bmatrix}.$$

Moreover, Table 4 states that $n_{\mathrm{x}_1} = 5$ and $n_{\mathrm{z}} = n_{\mathrm{z}_1} = 5$ for the GNSF model obtained by the transcription algorithm. This implies that α is indeed modeled through the LOS of GNSF within the automatically transcribed model. Note that as many states as possible are made part of the LOS, which means that the transcription principle 1 from Section 4.1 is fulfilled for this model.
The dynamics of α are stated in (5.3c) and are reformulated through the LOS (3.12b) which is defined by

$$E^{\mathrm{LO}} = [-1], \quad A^{\mathrm{LO}} = [0], \quad f_{\mathrm{LO}} = -v_\alpha.$$

The remaining model equations in (5.3) are reformulated within the NSF type system, which is defined by the following matrices

$$E = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 1 \end{bmatrix},$$

$$A = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}, \; C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \; c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 19.62 \end{bmatrix},$$

and the nonlinearity function

$$\phi(y, \hat{u}) = \begin{bmatrix} -3.5 - (F_{\mathrm{x}} + u)p_{\mathrm{y}} + F_{\mathrm{y}}p_{\mathrm{x}} \\ v_{\mathrm{y}}v_\alpha + p_{\mathrm{y}}a_\alpha \\ -(v_{\mathrm{x}}v_\alpha + p_{\mathrm{x}}a_\alpha) \end{bmatrix},$$

where $y = [\, p_{\mathrm{x}}, p_{\mathrm{y}}, v_{\mathrm{x}}, v_{\mathrm{y}}, v_\alpha, a_\alpha, F_{\mathrm{x}}, F_{\mathrm{y}} \,]^\top$ and $\hat{u} = u$ and the linear input matrices are defined such that $y = L_{\dot{\mathrm{x}}}\dot{x}^{[1]} + L_{\mathrm{x}}x^{[1]} + L_{\mathrm{z}}z^{[1]}$, $\hat{u} = L_{\mathrm{u}}u$, i.e.

$$L_{\mathrm{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad L_{\dot{\mathrm{x}}} = \mathbb{0}, \quad L_{\mathrm{z}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad L_{\mathrm{u}} = \begin{bmatrix} 1 \end{bmatrix}.$$

Note that only the states that are actually used within the ϕ expression are part of y ($\hat{u}$). Thus, we verified that transcription principle 3 from Section 4.1 is fulfilled for this model.

As stated in the previous chapter, the transcription algorithm sorts the model equations corresponding to the NSF part such that the first n_{x_1} equations correspond to $x^{[1]}$ and the following n_{z_1} equations correspond to $z^{[1]}$. In the NSF part of the GNSF model, the first five equations correspond to (5.3a), (5.3b) and (5.3d) to (5.3f), which means that the sorting worked for first n_{x_1} equations.

Regarding the Equations (5.3g) to (5.3k), it is not clear how to match the equations

and algebraic variables. The transcription method matched them as shown in Table 5.

Algebraic variable	a_{x}	a_{y}	a_{α}	F_{x}	F_{y}
Associated equation	(5.3j)	(5.3k)	(5.3i)	(5.3g)	(5.3h)

Table 5: Algebraic variables of the `inv_pend` model and equations associated with them by the automatic transcription method.

One can easily verify that the last five equations of the NSF type system are equivalent to the equations (5.3j), (5.3k), (5.3i), (5.3g) and (5.3h) in that order. This reformulation naturally results in a GNSF structured model that fulfills the requirement, i.e. the equations do not have to be modified within Algorithm 10.
For this particular model, we conclude that deploying the automatic transcription method resulted in a reasonable GNSF formulation that follows the essential principles in Section 4.1. Furthermore, the factor the dimension of the Newton matrix scales with within GNSF-IRK is $n_{\mathrm{out}} = 3$ instead of $n_{\mathrm{x}} + n_{\mathrm{z}} = 11$ for the standard implementation. Regarding the asymptotic complexity ratio in (3.62), we expect a speedup factor of $\left(\frac{11}{3}\right)^3 \approx 49$ within the linear system solves when comparing GNSF-IRK to the standard IRK implementation. Already having a look at Table 11, we observe that in practice this speedup is "only" 18.1 for $s = 7$, which is already a method of very high order. Thus, the complexity ratio in (3.62) should be considered with caution.

5.1.2 The `wt_nx13` Wind Turbine Model

We refer to the following model as `wt_nx13`, because it is a wind turbine model and consists of 13 differential states. The model is derived from a simplified flexible multibody formulation. The differential states of the model can be denoted as $x = \left[q^\top, \dot{q}^\top, \vartheta\right]^\top \in \mathbb{R}^{13}$, where 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 rotation angle of the generator and the collective deformation of all three blades in their first mode in tangential direction and in axial direction. The control vector $u = [T_{\mathrm{gen}}, \vartheta_{\mathrm{ref}}]^\top \in \mathbb{R}^2$ 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 (5.4).

$$0 = \begin{bmatrix} -3\ddot{q}_5 p_{19}\sin\vartheta + 3\ddot{q}_4 p_{18}\cos\vartheta + \ddot{q}_1 p_8 + \ddot{q}_1 p_7 + \ddot{q}_1 p_{28} + q_1 p_{27} + \dot{q}_1 p_{26} + 3\ddot{q}_1 p_{21} - F_{\text{thrust}} \\ p_1 T_{\text{rot}} + \ddot{q}_2 p_8 + \ddot{q}_2 p_7 + \ddot{q}_2 p_{29} + q_2 p_{27} + \dot{q}_2 p_{26} + 3\ddot{q}_2 p_{21} \\ -T_{\text{rot}} + 3\ddot{q}_4 p_{16}\sin\vartheta - 3\ddot{q}_5 p_{17}\cos\vartheta + (q_3 - q_6)\, p_6 + \ddot{q}_3 p_5 + 3\ddot{q}_3 p_{20} + (\dot{q}_3 - \dot{q}_6)\, p_2 \\ -p_{25}\sin\vartheta T_{\text{rot}} + 3\ddot{q}_3 p_{16}\sin\vartheta - F_{\text{thrust}} p_{23}\cos\vartheta + 3\ddot{q}_1 p_{18}\cos\vartheta + 3\ddot{q}_4 p_{14} + 3q_4 p_{12} + 3\dot{q}_4 p_{10} \\ p_{24}\cos\vartheta T_{\text{rot}} - F_{\text{thrust}} p_{22}\sin\vartheta - 3\ddot{q}_1 p_{19}\sin\vartheta - 3\ddot{q}_3 p_{17}\cos\vartheta + 3\ddot{q}_5 p_{15} + 3q_5 p_{13} + 3\dot{q}_5 p_{11} \\ p_9 T_{\text{gen}} + (\ddot{q}_6 p_4 + \ddot{q}_6 p_3)\, {p_9}^2 + (q_6 - q_3)\, p_6 + (\dot{q}_6 - \dot{q}_3)\, p_2 \end{bmatrix}, \tag{5.4}$$

where we use the shorthands

$$\begin{aligned} v_{\text{rot\_rel}} &= \dot{q}_1 + p_{30}\,\dot{q}_4\,\cos\vartheta, \\ v_{\text{wind\_eff}} &= v_{\text{wind}} - v_{\text{rot\_rel}}, \\ \lambda &= \dot{q}_3\, p_{33} / v_{\text{wind}} \end{aligned} \tag{5.5}$$

to define

$$\begin{aligned} P_{\text{rot}} &= 0.5 p_{32}\; {v_{\text{wind\_eff}}}^3\, c_{\text{p}}(\lambda, \vartheta) p_{31}, \\ T_{\text{rot}} &= P_{\text{rot}} \,/\, \dot{q}_3, \\ F_{\text{thrust}} &= 0.5 p_{32}\; {v_{\text{wind\_eff}}}^2\, c_{\text{t}}(\lambda, \vartheta). \end{aligned} \tag{5.6}$$

All parameter values p_i for $i = 1, \ldots, 33$ are confidential for the moment. The dynamics of ϑ read as

$$\dot{\vartheta} = (\vartheta_{\text{ref}} - \vartheta)/\tau_\vartheta. \tag{5.7}$$

Typically, in NMPC models for wind turbines, the aerodynamic coefficients $c_{\text{p}}, c_{\text{t}}$ are evaluated using splines [52]. However, within this model and the `wt_nx21` model presented in the next subsection, the aerodynamic coefficients are modeled as polynomials that are well defined only in a certain region, in which the simulations of our numerical experiments stay.

Since it was not possible to directly apply the automatic transcription method to wind turbine models that use splines, we chose to use the higher dimensional wind turbine models with the polynomial formulation, i.e. `wt_nx13` and `wt_nx21`.

The wind speed v_{wind} is a function parameter that is set to $12\frac{\text{m}}{\text{s}}$ for the simulation over a time horizon of 0.2s. We applied the automatic transcription method from Section 4.2 to the model equations and obtained an equivalent dynamic model in the GNSF form with the dimensions listed in Table 6.

Table 6: Dimensions of the `wt_nx13` model.

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

5.1.3 The `wt_nx21` Wind Turbine Model

We refer to the following model as `wt_nx21`, because it is a wind turbine model and consists of 21 differential states. The model is very similar to the `wt_nx13` model. The difference compared to the `wt_nx13` model is that $q \in \mathbb{R}^{10}$, where the first four components of q have the same purpose as for the `wt_nx13` model. However, the last six components of q model the three blades in their first mode in tangential direction and in axial direction separately. Within the `wt_nx13` model, these quantities are modeled collectively using only two components of q. This explains why the factor 3 occurs so often in (5.4). The equation describing the behavior of $\ddot{q}$ can be found in Equation (6.1) in the appendix.
We used the automatic transcription method from Section 4.2 to obtain an equivalent model in the GNSF structure. The dimension of the implicit and the equivalent GNSF model can be found in Table 7.

Table 7: Dimensions of the `wt_nx21` model.

Model dimensions				GNSF dimensions				
n_{x}	n_{u}	n_{z}	n_{param}	n_{x_1}	n_{z_1}	n_{out}	n_{y}	$n_{\hat{\mathrm{u}}}$
21	2	0	1	21	0	10	27	1

5.1.4 The `wt_nx6` Wind Turbine Model

We will refer to this model as `wt_nx6`, as it is a wind turbine model and has six differential states. In this model, the aerodynamic coefficients $c_{\mathrm{p}}, c_{\mathrm{t}}$ are modeled by splines. The splines used within this model are relatively costly to evaluate. Although, a fast spline evaluation was developed recently, it not yet available in `CasADi` [53]. Because of the spline formulation that can only be used with `CasADi MX` symbolics, the model was transcribed by hand instead of using the automatic transcription algorithm. The resulting model dimensions can be found in Table 8.
The model is also used in the `acados C` examples and can be found in the folder `/examples/c/wt_model_nx6`.

Table 8: Dimensions of the `wt_nx6` model.

Model dimensions				GNSF dimensions				
n_x	n_u	n_z	n_param	n_{x_1}	n_{z_1}	n_out	n_y	$n_{\hat{\mathrm{u}}}$
6	2	0	0	6	0	1	5	0

5.2 Comparing Computation Times

In this section, we want to get some insight on how GNSF-IRK can be compared with the standard IRK implementation regarding the distribution of computation time spent within the algorithms. Since there are different approaches to compare computation times, we first want to discuss them briefly.

In [47] a runtime comparison was done for the NSF exploiting IRK. There, the authors chose to compare the total time (*tot*) spent in the integrator and the time spent to compute linear system solutions (*lss*), i.e. matrix factorizations and triangular solves. In this section, we additionally regard the time spent in linear algebra within the integrator (*la*), which is regarded as the most important timing in the comparison. First, this is due to the fact that the number of calls to and time spent inside external function evaluations is close to constant for all IRK implementation. Second, the speedup within this linear algebra time can be regarded as the actual speedup in the integration scheme regarding Amdahl's law, since only these operations are changed by the structure exploitation.

If the Jacobians are not reused, the number of integration steps n_steps has rather small influence on the timing comparison. Therefore, in the numerical experiments of this section, we use $n_\mathrm{steps} = 1$. Both algorithms are used with $n_\mathrm{newton} = 3$, Butcher tableaux corresponding to the Gauss-Legendre collocation method and the option to propagate first order forward sensitivities.

Figure 4 illustrates the values in Table 9, in which also the speedup of GNSF-IRK w.r.t. the standard IRK implementation is listed. We observe that the time spent to evaluate the external functions is not significantly different in the IRK implementations. However, the time required to solve the linear systems and to carry out the remaining operations is strongly dependent on the IRK implementation. For the standard IRK implementation, the lss time is indeed growing fast with the number of stages. For $s = 2$ nearly half of the CPU time is spend to solve the linear systems and for very high orders ($s = 7$), solving the linear systems takes more than 85 % of the CPU time.

We see that GNSF-IRK is indeed mainly developed with the intention to reduce this (lss) time. The speedup regarding just the linear system solutions (lss) ranges from 3 to more than 8.

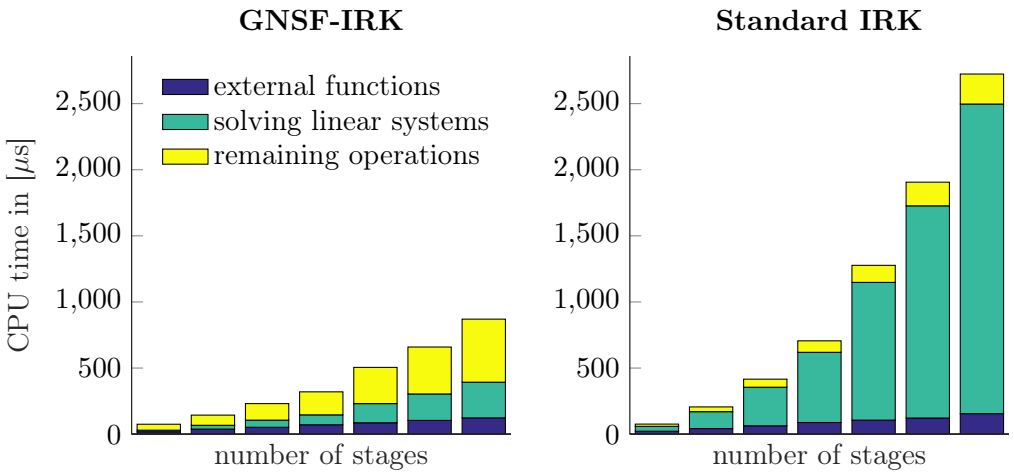

Figure 4: Comparison of computation times for standard IRK and GNSF-IRK, using the `wt_nx13` model. The values presented can be found in Table 9.

s	GNSF-IRK			Standard IRK			Speedup Factor		
	tot	la	lss	tot	la	lss	tot	la	lss
1	75	57	12	76	54	37	1.01	0.947	3.08
2	144	106	29	206	164	128	1.43	1.55	4.41
3	231	179	54	416	353	292	1.8	1.97	5.41
4	320	249	75	706	618	531	2.21	2.48	7.08
5	505	419	145	1280	1170	1040	2.53	2.79	7.19
6	659	555	200	1910	1780	1610	2.89	3.21	8.03
7	870	747	270	2720	2570	2340	3.13	3.44	8.67

Table 9: Comparison of computation times for standard IRK and GNSF-IRK, using the `wt_nx13` model. Timings are in [μs], factors have no unit.

Regarding the remaining operations, which are mainly matrix-vector and matrix-matrix multiplications, one observes that there are much more operations needed when using the structure exploiting implementation. For $s = 1$, these operations take more than three times the time to solve the linear systems, (the ratio $\frac{\text{la}-\text{lss}}{\text{lss}}$). However, as the matrices are of lower dimension, the complexity of these operations still grows much slower than the complexity of solving the linear systems. For GNSF-IRK with $s = 7$, the remaining operations still take around 1.7 times the lss time, so asymptotically they are still cheaper compared to the linear solves.

Regarding Table 9, one observes that all listed speedups grow with the number of stages s, corresponding to the order of the method. While the total speedup is negligible for $s = 1$, it is greater than two for $s = 4$ and greater than three for $s = 7$. In the case $s = 1$, there is a total speedup of 1%. However, considering only the time to perform linear algebra operations within the integrator, GNSF-IRK is 5% slower.

Thus, there must be a speedup within the external function calls, which is due to the fact, that some linear operations are removed from the code generated function ϕ. Note that the speedup when comparing the linear algebra operations, is very close to the total speedup, as the external function calls are rather cheap for this model. In order to give an intuition that this is not always the case, let us regard the `wt_nx6` model.

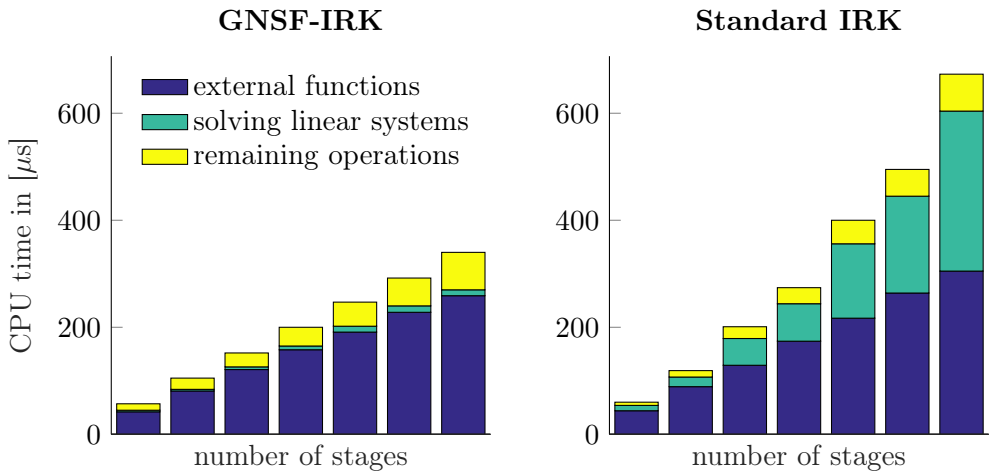

Figure 5: Comparison of computation times for standard IRK and GNSF-IRK, using the `wt_nx6` model. The values presented can be found in Table 10.

s	GNSF-IRK			Standard IRK			Speedup Factor		
	tot	la	lss	tot	la	lss	tot	la	lss
1	57	15	3	60	16	10	1.05	1.07	3.33
2	105	24	3	119	30	18	1.13	1.25	6
3	152	31	5	201	72	50	1.32	2.32	10
4	200	42	7	274	100	70	1.37	2.38	10
5	247	56	11	400	183	139	1.62	3.27	12.6
6	292	64	12	495	231	181	1.7	3.61	15.1
7	340	81	11	673	368	299	1.98	4.54	27.2

Table 10: Comparison of computation times for standard IRK and GNSF-IRK, using the `wt_nx6` model. Timings are in [μs], factors have no unit.

In Figure 5 and Table 10 the same kind of comparison is done for the `wt_nx6` model. Here, the external function calls take a lot of time, which is due to the fact that an inefficient spline implementation is used to model the aerodynamic coefficients. Regarding $s = 7$ and the ratio $\frac{\text{tot}-\text{la}}{\text{tot}}$, we find that 76% of GNSF-IRKs computation time is spent in the external functions, whereas it is only 45% for the standard implementation.

However, GNSF-IRK still provides a significant total speedup, ranging from 5% to 98%. Considering the linear algebra timing (la), GNSF-IRK is up to 4.5 times faster compared to the standard implementation. Thus, one could argue that this model is even better suited for GNSF-IRK compared to the `wt_nx13` model. This can be justified by the ratio $\frac{n_{\text{out}}}{n_{\text{x}}}$, which is $\frac{1}{6} \approx 0.1667$ for `wt_nx6` compared to $\frac{5}{13} \approx 0.3846$ for `wt_nx13`. Hence, one can expect an even higher total speedup for this model when using an efficient spline implementation.

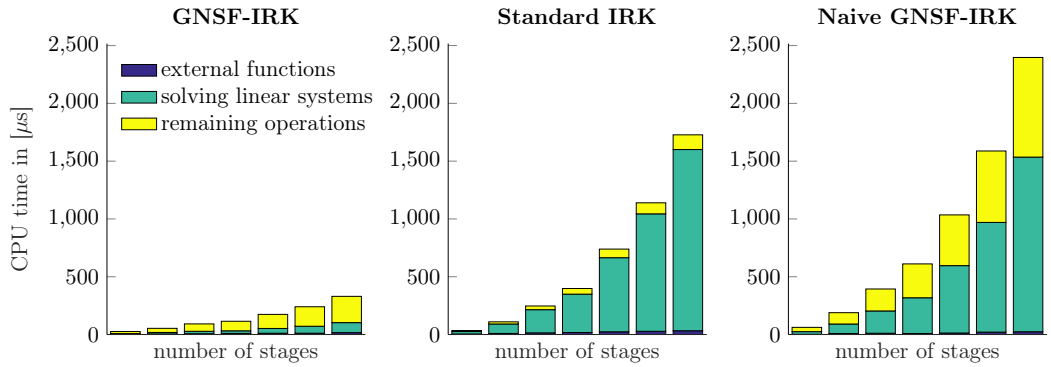

Figure 6: Comparison of computation times for standard IRK and GNSF-IRK, using the `inv_pend` model with automatic GNSF transcription Table 4 in the left subplot and Table 11. In the right subplot, a naively transcribed GNSF model is used, see Table 12 and Table 13.

s	GNSF-IRK			Standard IRK			Speedup Factor		
	tot	la	lss	tot	la	lss	tot	la	lss
1	26	23.3	5.2	34.7	29.8	21	1.33	1.28	4.04
2	53.6	48.2	12.7	110	100	81.8	2.05	2.08	6.44
3	91.9	85	21	247	233	201	2.69	2.74	9.56
4	115	105	22.7	398	381	332	3.47	3.62	14.6
5	174	163	41.5	739	715	641	4.24	4.38	15.4
6	240	229	60.2	1140	1110	1020	4.74	4.86	16.9
7	330	314	86.6	1730	1690	1570	5.23	5.4	18.1

Table 11: Comparison of computation times for standard IRK and GNSF-IRK, using the `inv_pend` model with the automatic transcription method, Table 4. Timings are in [μs], factors have no unit.

For the `inv_pend` model, a similar timing comparison is shown in the first two subplots of Figure 6 and Table 11.
The additional subplot on the right side shows the computation times for a naively transcribed GNSF model. This reformulation was obtained using the transcription

method in Chapter 4 but skipping the essential steps in lines 16 to 20 in Algorithm 7. The dimensions of this naive GNSF reformulation are listed in Table 12. The timings of standard IRK and GNSF-IRK with the naive reformulation are listed in Table 13. For this model, GNSF-IRK gives a really good speedup considering the total time. It is greater than two for $s \geq 2$ and greater than four for $s \geq 5$.

Table 12: Dimensions of the inverted pendulum (`inv_pend`) model with a *naively transcribed GNSF model*, compare to Table 4.

Model dimensions				GNSF dimensions				
n_x	n_u	n_z	n_param	n_{x_1}	n_{z_1}	n_out	n_y	$n_{\hat{\mathrm{u}}}$
6	1	5	0	6	5	11	8	1

s	GNSF-IRK			Standard IRK			Speedup Factor		
	tot	la	lss	tot	la	lss	tot	la	lss
1	62	60	23	36	30	23	0.581	0.5	1
2	189	180	82	112	102	84	0.593	0.567	1.02
3	394	383	193	249	237	202	0.632	0.619	1.05
4	611	602	309	398	378	331	0.651	0.628	1.07
5	1040	1020	583	710	687	615	0.686	0.672	1.05
6	1590	1570	949	1120	1090	995	0.704	0.696	1.05
7	2400	2370	1510	1670	1640	1520	0.696	0.692	1

Table 13: Comparison of computation times for standard IRK and GNSF-IRK, using the `inv_pend` model with a *naive* GNSF reformulation, Table 12. Timings are in [μs], factors have no unit.

In contrast to this, transcribing a model naively into the GNSF format results in a significantly longer runtime of GNSF-IRK compared to the standard IRK implementation. Since the linear systems for both algorithms are of the same dimension, namely $n_\mathrm{out} = n_\mathrm{x} + n_\mathrm{z}$, the linear solves take approximately the same time, see Table 13. Moreover, the remaining operations are much more expensive for the naively transcribed GNSF model, because many of the matrix multiplications have a dimension depending on n_out.

This comparison was mainly made to show both, the necessity of having an algorithm to obtain a suitable GNSF version of the model, and the possible overhead that can arise in the application of GNSF-IRK to an inappropriate model, i.e. one with $n_\mathrm{x} + n_\mathrm{z} \approx n_\mathrm{out}$.

5.3 Convergence and Initialization

In this section, GNSF-IRK and the standard IRK algorithm are compared regarding their convergence properties and natural initialization.
As both schemes are designed to solve a mathematical equivalent system of equations, the converged solutions should always be the same. However, the numerical properties of the schemes are not the same as Newton iterations are performed on a different space. Moreover, the natural initialization, i.e. initializing the integration variables (K, respectively $\mathbf{v}$) with zeros, distinguishes the two schemes.
In order to compare the behaviour of the algorithms, the standard IRK integrator should be initialized equivalently to the natural initialization of the GNSF-IRK. Initial values for standard IRK that are equivalent to the natural initialization of GNSF-IRK can be easily obtained by first computing

$$\begin{bmatrix} \dot{x}_0^{[1]} \\ z_0^{[1]} \end{bmatrix} = E^{-1}(Ax_0^{[1]} + Bu_0 + c), \tag{5.8}$$

and subsequently

$$\begin{bmatrix} \dot{x}_0^{[2]} \\ z_0^{[2]} \end{bmatrix} = \left(E^{\mathrm{LO}}\right)^{-1}(A^{\mathrm{LO}}x_0^{[2]} + f_{\mathrm{LO}}(\dot{x}_0^{[1]}, x_0^{[1]}, z_0^{[1]}, u_0)), \tag{5.9}$$

where x_0 denotes the initial state and u_0 the applied control. Now, the standard IRK scheme can be initialized equivalently to GNSF using $k_i = \dot{x}_0$ and $z_i = z_0$, for $i = 1, \ldots, s$.
First, we want to discuss Figure 7 in detail, as it is a typical plot for this section. It shows the behavior of our IRK implementations, i.e. standard and GNSF, within the first Newton iterations applied to an IVP from Section 5.1, here the `inv_pend` model. The results for the other models are presented in the following figures. The standard IRK is listed twice, with its natural initialization (IRK: zero-init) and with the one equivalent to the GNSF-IRK initialization (IRK: gnsf-init), that is described above. For $n_{\mathrm{newton}} = 3$, there is hardly a difference concerning accuracy, as one can see in Section 5.4. Thus, we only regard the first two Newton iterations in this section.
In Figure 7, we observe that for the `inv_pend` model GNSF-IRK has the same convergence behavior as the standard IRK with the equivalent initialization. Additionally, we note that the natural IRK initialization is significantly worse compared to the one of GNSF-IRK.
In Figure 8, only the first Newton iteration applied to the `wt_nx6` model is presented because for this model, we can hardly observe a difference between the methods.

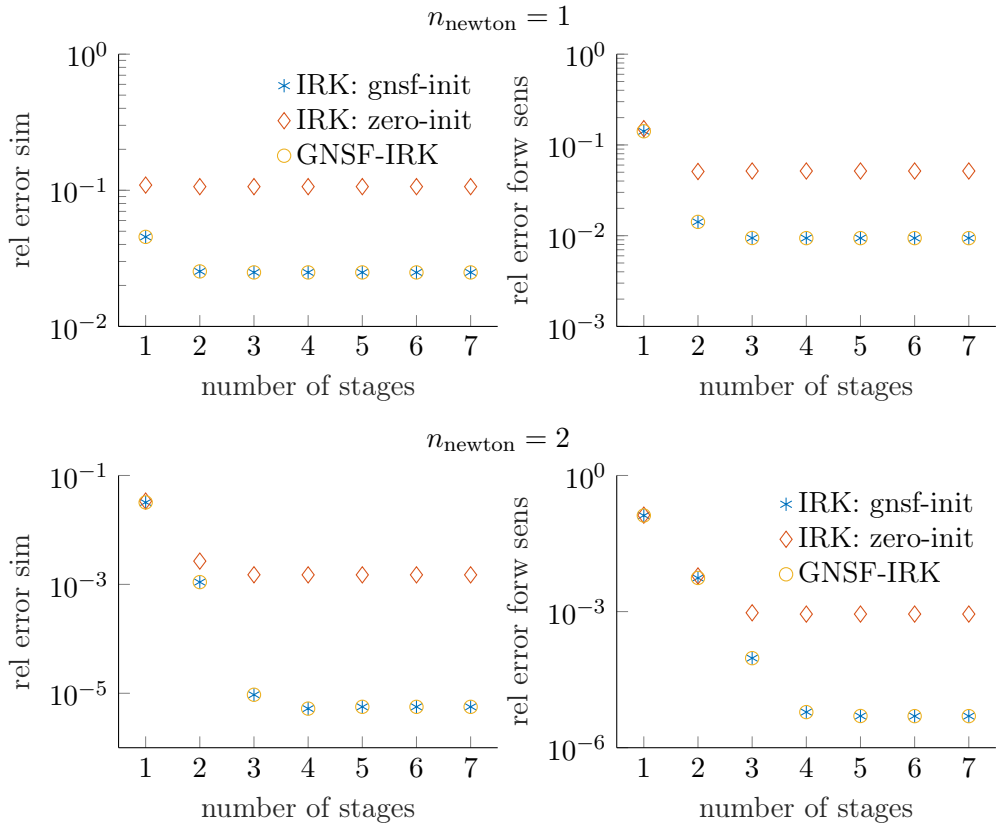

Figure 7: `inv_pend` model: Error comparison standard IRK vs. GNSF-IRK.

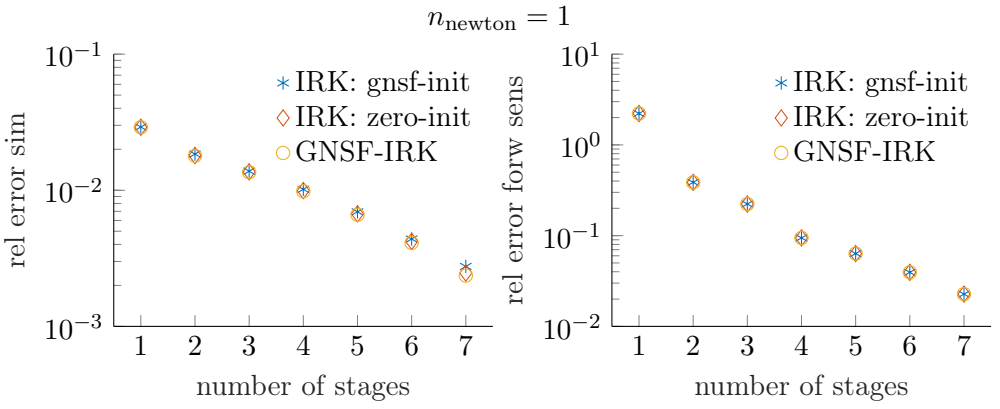

Figure 8: `wt_nx6` model: Error comparison standard IRK vs. GNSF-IRK.

Actually, there is only a slight difference visible between the IRK initializations with $s = 7$. In this case, the standard IRK implementation with the GNSF initialization is marginally more accurate compared to the other algorithms.

The results for the `wt_nx13` and the `wt_nx21` model are presented in Figure 9 and Figure 10. Since there is hardly a difference between these figures observable, we want to discuss them simultaneously. After a single Newton iteration, the standard IRK implementation with the GNSF initialization provides the best solution, followed

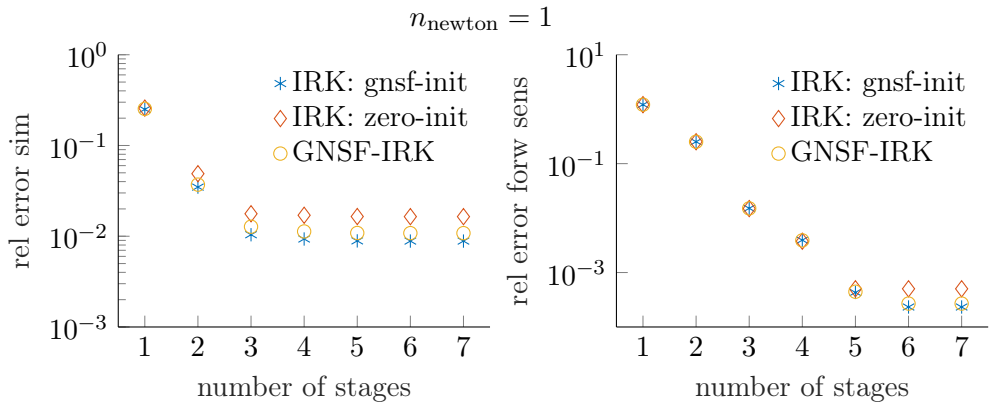

Figure 9: `wt_nx13` model: Error comparison standard IRK vs. GNSF-IRK.

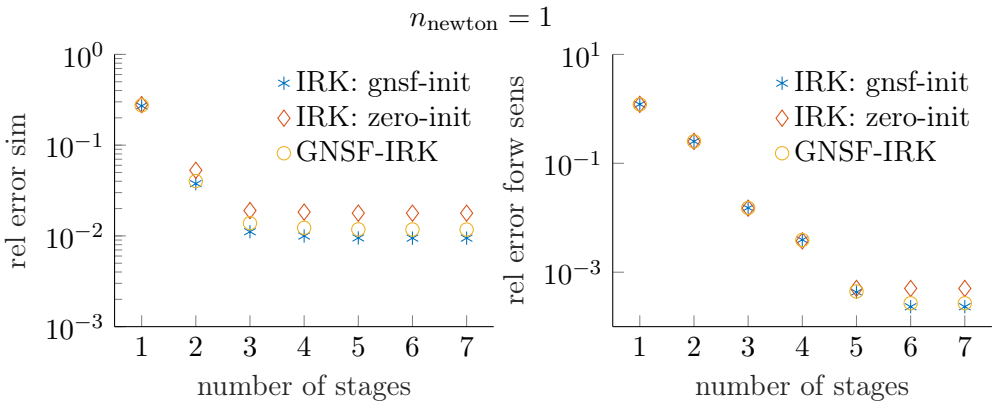

Figure 10: `wt_nx21` model: Error comparison standard IRK vs. GNSF-IRK.

by GNSF-IRK and the standard IRK implementation with its natural initialization. The difference between the methods is bigger if more stages are used. However, the difference between all methods is relatively small, smaller than 0.05. For $n_{\text{newton}} = 2$, there is no difference visible and the corresponding plots are omitted.

The observations in this section can be summarized as follows. For the test problems used, the GNSF initialization is better or equally good compared to the natural standard IRK initialization. An intuitive explanation for this is that the GNSF initialization takes the linear dependencies into account and neglects only the nonlinear term ϕ, respectively the GNSF-IRK initialization assumes ϕ to be zero. However, this is just a heuristic and one can easily come up with counter examples, in which the standard IRK initialization is favorable.
On the other hand, we saw that for two test problems, the `wt_nx13` and the `wt_nx21` model, the standard IRK with GNSF initialization has a slightly better convergence rate compared to GNSF-IRK. One possible explanation for this is that some matrices used within GNSF-IRK are not as well conditioned, which results

in less accurate solutions of the linear systems. This could be investigated further, and one could modify the automatic GNSF transcription aiming for well conditioned matrices.
Another reason for the above results could be that within GNSF-IRK the integration equations corresponding to the linear dynamics are solved separately and exactly, using the current solution for the nonlinear equations. In contrast to that, standard IRK Newton iterations solve both linear and nonlinear equations simultaneously.
Finally, we state that the convergence properties observed in this section are similar to the ones of the standard IRK implementation and even if they are slightly worse, the saving of computation time compensates for this, see Section 5.2 and Section 5.4.

5.4 Efficiency Comparison

In this section, GNSF-IRK is compared with the standard IRK algorithm considering accuracy and computation time together in order to analyze their overall efficiency. Therefore, we consider different numbers of steps n_{steps} and numbers of stages s.
For the numerical experiments discussed in the remainder of this thesis, it was chosen to always use all possible combinations of the parameter values shown in Table 14. Moreover, we always simulate and propagate the first order forward sensitivities. However, in the plots of this section, we restrict ourselves to $n_{\text{newton}} = 3$, as for this value the different initializations of the algorithms, discussed in Section 5.3, have only slight effects on the results.

n_{steps}	s	n_{newton}
1	1	1
2	2	2
3	3	3
4	4	4
5	5	
7	6	
10	7	
20		
50		
100		

Table 14: Parameters of IRK schemes used within the experiments presented in this chapter.

In the numerical experiments, the relative error of the simulation and sensitivity propagation is computed with respect to a reference solution that is obtained using an IRK method with $s = 8$, $n_{\text{newton}} = 5$ and $n_{\text{steps}} = 400$.
The plots in this section are meant to give a measure for efficiency by plotting

the relation between the relative error and the CPU time using a logarithmic scale for both axes. The results for different values of n_{steps} and the same value for s are presented as one polygonal line within the plot. Since it is possible to use the integration methods with different values for $n_{\text{steps}} \in \mathbb{N}$ apart from the ones used to generate the plot and all step sizes $h > 0$ by varying T, these lines are meaningful and allow for a relatively fair comparison of the methods. Similar plots, in which the axis corresponding to the error is reversed, are called work-precision diagrams [31]. Additionally, the lower left part of all the lines plotted can be regarded as the efficient set, also called Pareto-front, of the biobjective optimization problem that one would naturally formulate to choose an integration method by taking the computation time and the accuracy as objective functions. This property is discussed more precisely in Section 5.5.

As the natural initialization of GNSF and its strong effects on the results in the

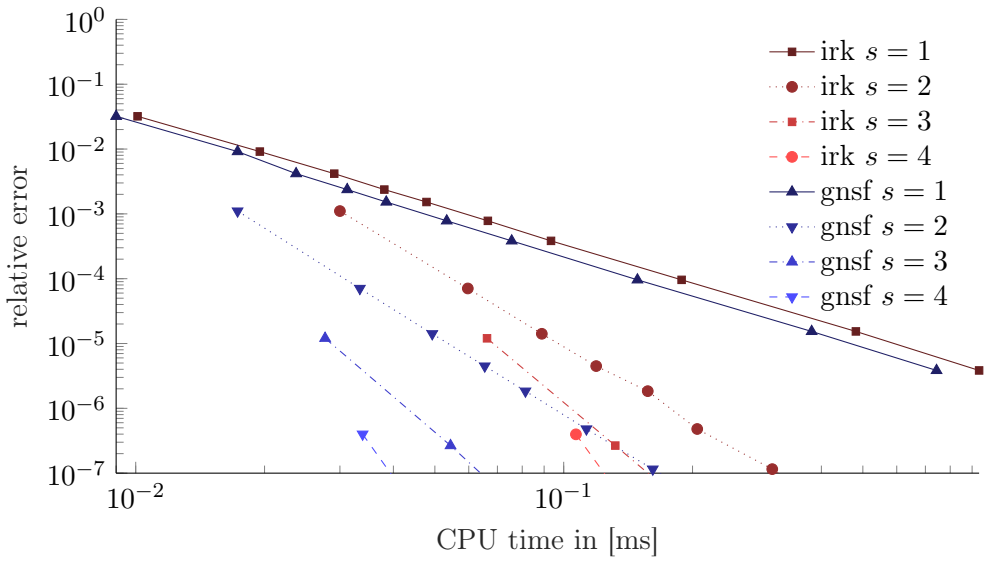

Figure 11: `inv_pend` model: Different orders of IRK using GNSF-IRK and the standard IRK implementation respectively, for $n_{\text{newton}} = 3$, with natural initialization. The values for n_{steps} are listed in Table 14.

first Newton iterations are discussed in Section 5.3, this section will focus on a number of Newton iterations that is typical within a multiple shooting algorithm, say $n_{\text{newton}} = 3$, which is the default in `acados`. For every IVP, it has to be investigated individually, which parameter values, Table 14, are suitable to solve it efficiently.

Both IRK implementations were applied to the `inv_pend` model and the results are visualized in Figure 11, which is a typical type of plot for this section. After three Newton iterations, typically both IRK implementations converge to the same solution. Thus, the lines of GNSF-IRK and the standard IRK that use the same s and Butcher

tableau, can be easily identified with each other. They provide the same, or not distinguishable, accuracy for the same number of steps such that these plots can be seen as a proof of work. Note that this holds even though both implementations are used with their natural initialization (Section 5.3). However, as they have different CPU times, it looks like the corresponding lines are shifted in x-direction. The length of this shift corresponds to the speedup provided by GNSF-IRK with respect to the standard implementation. Note that for higher order methods, the shifts and speedups are bigger, which we have already seen in Section 5.2. As the 4 stage method with one integration step provides a very good accuracy, the methods with $s > 4$ are not shown in Figure 11.

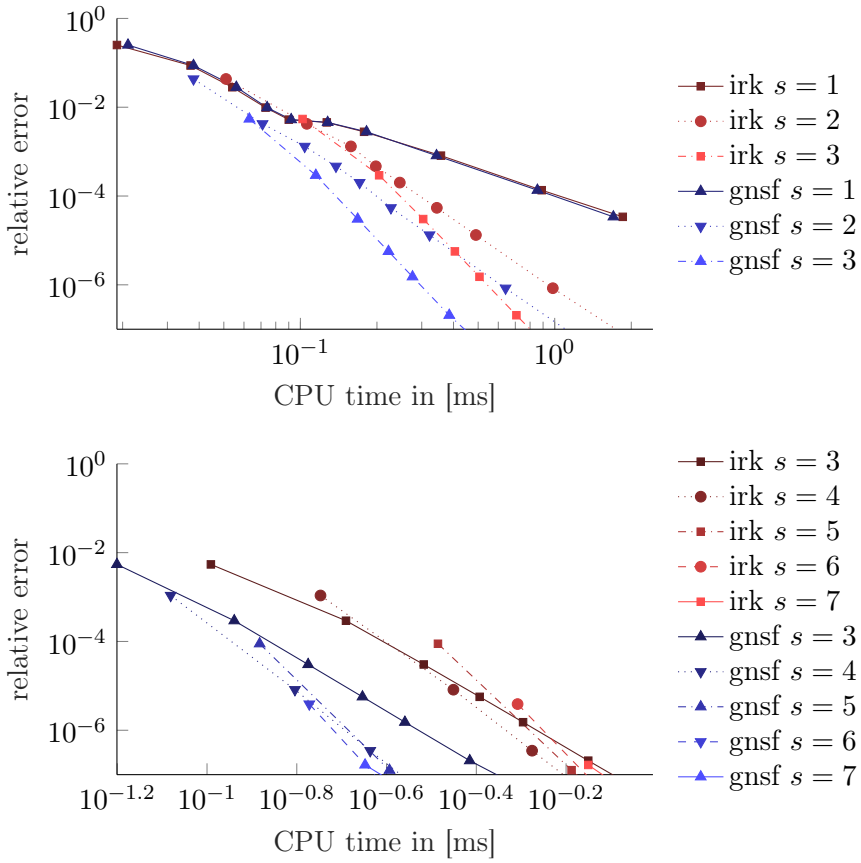

Figure 12: `wt_nx13` model: Different orders of IRK using GNSF-IRK and the standard IRK implementation respectively, for $n_{\text{newton}} = 3$, with natural initialization. The values for n_{steps} are listed in Table 14.

Figure 12 shows a similar comparison for the `wt_nx13` model. It is separated into two subplots to improve readability. In the first subplot $s = 1, 2, 3$ are shown and in the second one the methods with $s = 3, \ldots, 7$ are visualized.

Regarding the lines for $s = 1$, one observes that the CPU time for GNSF-IRK grows a bit slower with the number of steps. This can be reasoned by the fact that there are some operations that have to be performed just once in GNSF-IRK independently of the number of steps. This effect can be found for all models, i.e. also in Figure 11, Figure 13 and Figure 14.
The lines for $s = 1$ also show how well-suited the model is for the structure exploitation, meaning how far the dimensions n_{out} and n_{y} are reduced with respect to their upper bound $n_{\text{x}} + n_{\text{z}}$ and $2n_{\text{x}} + n_{\text{z}}$ respectively. For the `inv_pend` and the `wt_nx6` model GNSF-IRK is always faster compared to the standard implementation. In contrast to that, for the `wt_nx13` and the `wt_nx21` model, GNSF-IRK is only faster if a minimum number of steps is used.

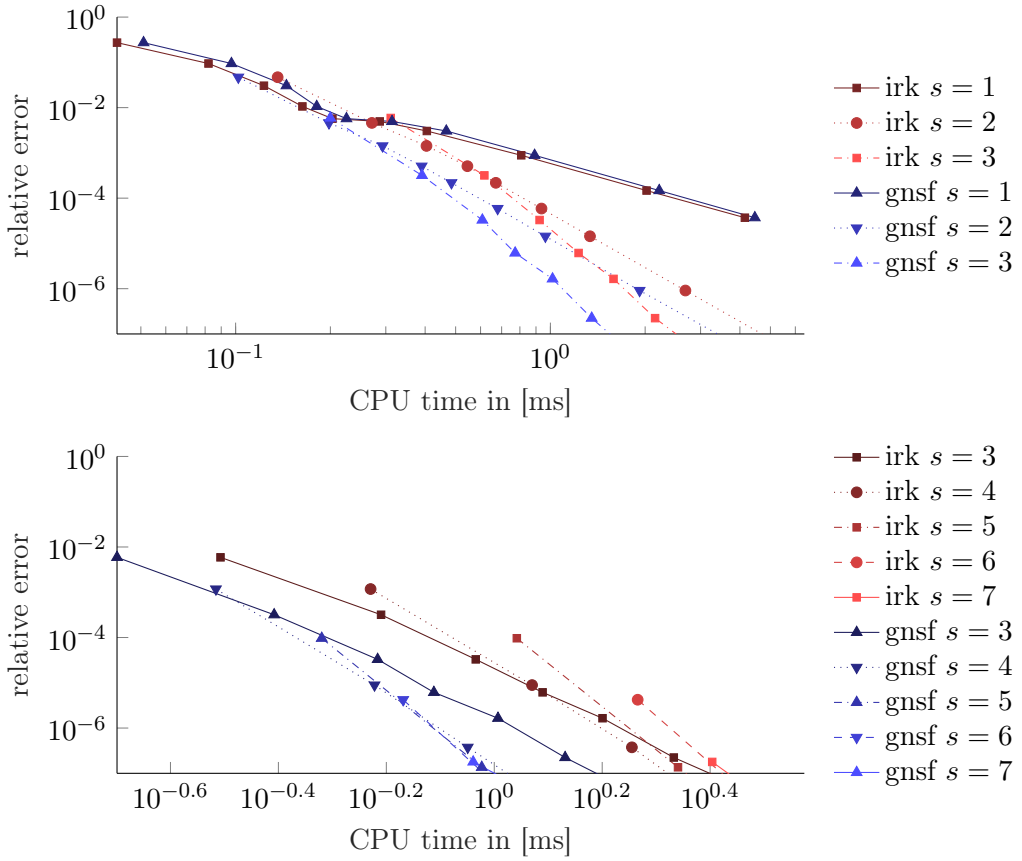

Figure 13: `wt_nx21` model: Different orders of IRK using GNSF-IRK and the standard IRK implementation respectively, for $n_{\text{newton}} = 3$, with natural initialization. The values for n_{steps} are listed in Table 14.

In the second subplot in Figure 12, the results of the IRK implementations are separated, because GNSF-IRK is significantly faster for higher order methods.

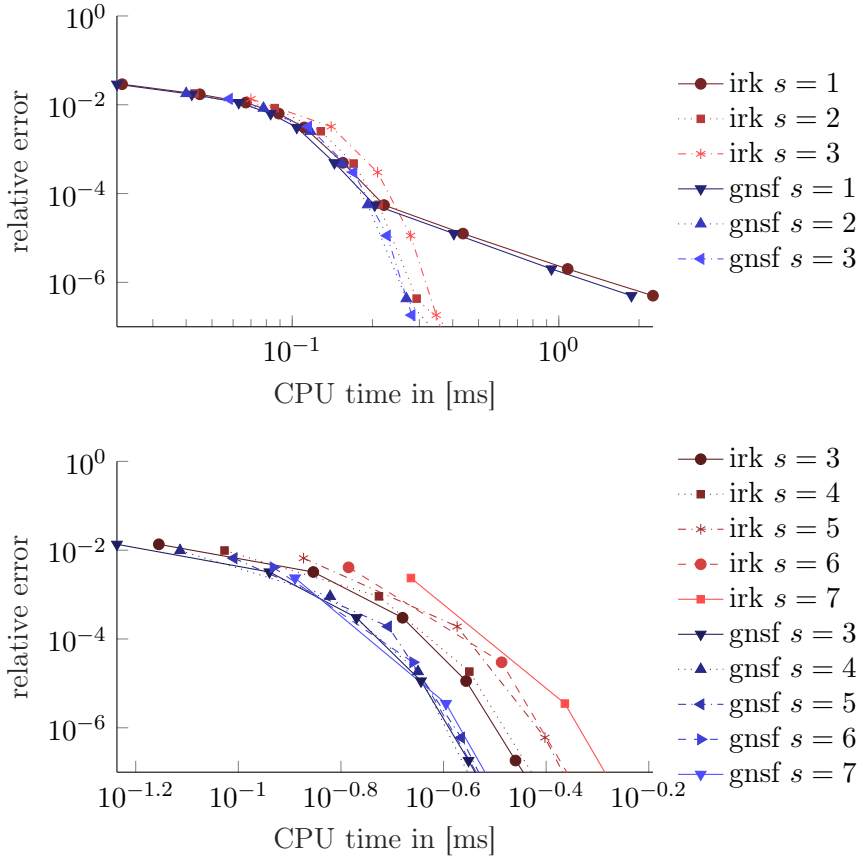

Figure 14: `wt_nx6` model: Different orders of IRK using GNSF-IRK and the standard IRK implementation respectively, for $n_{\text{newton}} = 3$, with natural initialization. The values for n_{steps} are listed in Table 14.

Let us take a closer look at the efficient set of methods. For GNSF-IRK, this efficient set contains methods with all values for s, i.e. 1 to 7. In contrast to this, the efficient set of the standard implementation only contains methods with values for s ranging from 1 to 5. Thus, we conclude that the optimal choice of the IRK settings, meaning the Butcher tableau and number of integration steps n_{steps}, strongly depends on which IRK implementation (GNSF or standard) is used. The availability of a structure exploiting IRK scheme makes it more attractive to use higher order methods with a lower number of integration steps. This qualitative result can also be seen regarding the results for the `wt_nx21` and `wt_nx6` model in Figure 13 and Figure 14.

For a truly fair timing comparison of the schemes, one should compare only the methods that belong to the efficient set of settings of an IRK algorithm. Namely, one should compare higher order methods of GNSF-IRK with lower order methods of the standard IRK that use more integration steps and thus provide a similar accuracy.

5.5 Efficiency Comparison with Pareto front

For the sake of completeness, we formally define some terms from the field of multiobjective optimization formally in the following, using the notation in [54].

Definition 5.1 Given a nonempty set $S \subseteq \mathbb{R}^n$ and a function $f : S \to \mathbb{R}^m$, the problem

$$\underset{x \in S}{\text{minimize}} \quad f(x) \tag{5.10}$$

is called *multiobjective* or *multicriteria optimization problem*. In the case of $m = 2$, (5.10) is also called *biobjective* optimization problem.
A point $\bar{x} \in S$ is called Edgeworth-Pareto optimal (EP optimal) solution of (5.10) if there exists no $x \in S \setminus \{\bar{x}\}$ such that

$$\begin{aligned} & f_i(x) \leq f_i(\bar{x}) \quad \text{for } i = 1, \ldots, m, \\ \text{and} \quad & f_j(x) < f_j(\bar{x}) \quad \text{for at least one } j \in \{1, \ldots, m\}. \end{aligned}$$

The set of images of optimal solutions of a multiobjective optimization problem, i.e.

$$\mathcal{E} = \{\, y = f(x) \in \mathbb{R}^m \mid x \text{ is an EP optimal solution of (5.10)} \,\},$$

is called *efficient set*, also Pareto front or Pareto set [55].

We use the above definitions to visualize and evaluate the numerical results from the previous section further.
In Figure 15, all EP optimal methods for both IRK algorithms are highlighted and connected by a stair graph. In this setting, it makes sense to use the stairs that "underestimate" the efficient set, which can be interpreted as follows.
The stair plot can be evaluated at a specific time, the maximal amount of time in which the integrator must be able to carry out its tasks. The function value of the stairs provides the best accuracy that can be obtained by this IRK algorithm within the given time.
Additionally, one can switch the axes of the plot to obtain a very similar plot, in which the stair plot is still well-defined. Here, we use the convention that the function value of the stairs plot at a point of discontinuity is defined as the minimum of the vertical line. Given a specific accuracy that at least should be satisfied by the integrator, we evaluate the stair plot for this accuracy and obtain the minimal time this integrator needs to provide at least this accuracy.

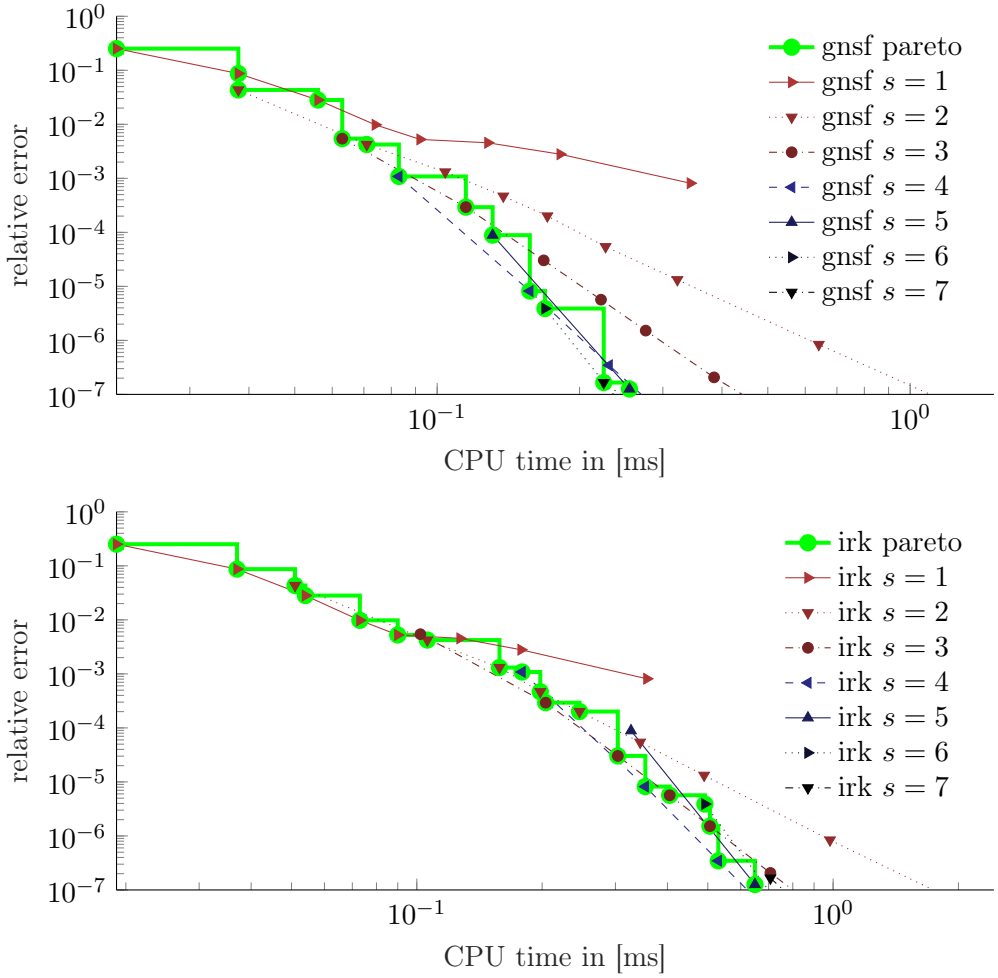

Figure 15: `wt_nx13` model: Different order IRK methods computed with GNSF-IRK and standard IRK. The EP-optimal methods are highlighted and connected by stairs as an approximation of the efficient set.

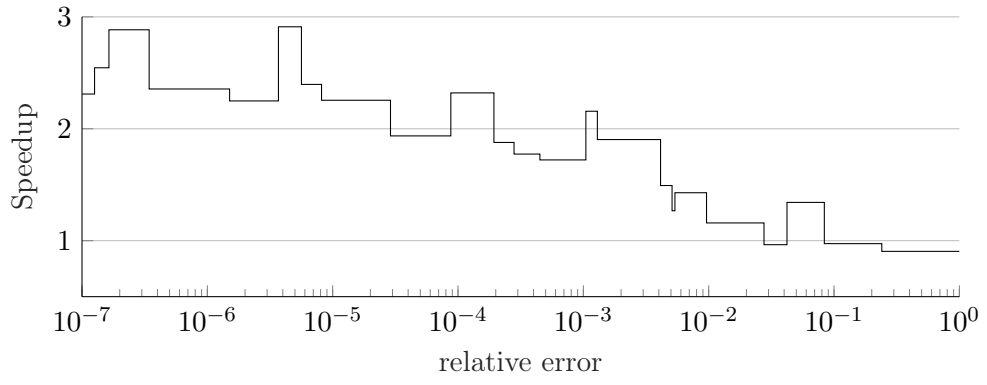

Figure 16: `wt_nx13` model: Speedup of GNSF-IRK w.r.t. the standard IRK, using the highlighted approximation of the efficient set in Figure 15.

The computation times that are at least needed to provide a certain accuracy with GNSF-IRK and the standard implementation can then be compared to get the speedup of GNSF-IRK w.r.t. standard IRK when choosing the settings for both algorithms efficiently. Figure 16 visualizes how this speedup is behaving for varying required accuracies using the `wt_nx13` test problem.

Taking a closer look at Figure 15, we can now specify which methods are EP-optimal for GNSF-IRK and standard IRK respectively. Starting with low accuracies, the first difference is that the method with $s = 1$ and $n_{\text{steps}} = 4$ is EP-optimal for the standard IRK but not for GNSF-IRK. The method with $s = 3$ and $n_{\text{steps}} = 1$ is EP-optimal for GNSF-IRK, but not for the standard IRK implementation. Whereas, the methods with $s = 3$ and $n_{\text{steps}} = 3, 4, 5$ are EP-optimal for standard IRK, but not for GNSF-IRK. However, we already found out in the last section that for GNSF-IRK higher order methods are more favorable even for relatively low accuracies. Thus, this kind of plots does not provide new qualitative information, and they are omitted for the other models.

Considering Figure 16, we observe that the speedup of GNSF-IRK compared to the standard IRK is especially good for high accuracies. Regarding low accuracies, i.e. where the relative error is greater than 10^{-2}, GNSF-IRK is partly 5% to 10% slower compared to the standard implementation. Whereas for higher accuracies a speedup factor of up to 2.9 can be observed.

Within an NMPC scheme, the accuracy that should be satisfied by an integrator is strongly depending on the practical application and the model. Sometimes medium accuracy is fine because there is an uncertainty within the model or noise. Additionally, when controlling systems with fast dynamics, one might not afford computing very accurate solutions [56]. In the medium to low accuracy range, i.e. relative error between 10^{-4} and 10^{-2}, a speedup between 1.5 and 2 can be expected using the `wt_nx13` model. On the other hand, for some applications, the sampling times and time horizon are relatively long and there exist well-engineered models. For example, regarding wind turbines, in [57], [52] sampling times of 0.2s and 0.1s are used respectively. Thus, it could make sense to use a very accurate integrator with a relative error smaller than 10^{-4}. In this case, a speedup greater than 2 seems to be a realistic estimation.

Similar plots are presented in Figure 17, Figure 18 and Figure 19 for the other test problems.

Figure 17 shows again, that the `inv_pend` model is very well suited for GNSF-IRK. Given any accuracy, one can obtain a positive speedup by using GNSF-IRK. For accuracies better than 10^{-3}, the speedup factor is always greater than 2 for accuracies better than 10^{-5} it is mostly greater than 3.

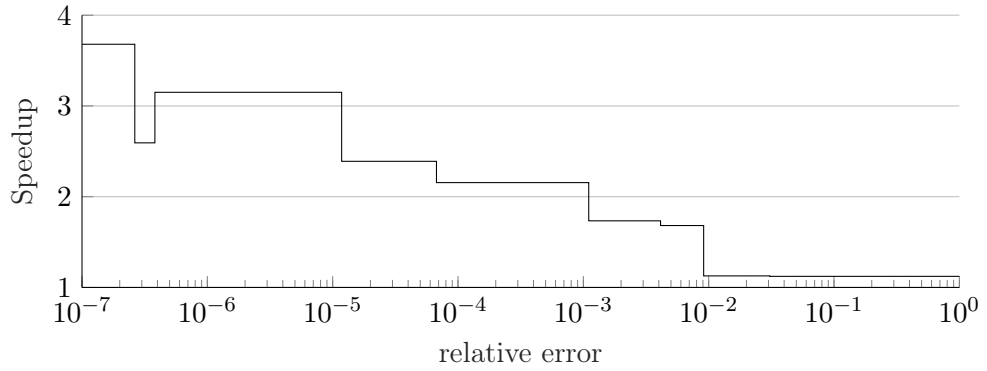

Figure 17: `inv_pend` model: Speedup of GNSF-IRK w.r.t. the standard IRK, using an approximation of the efficient set similar to Figure 15.

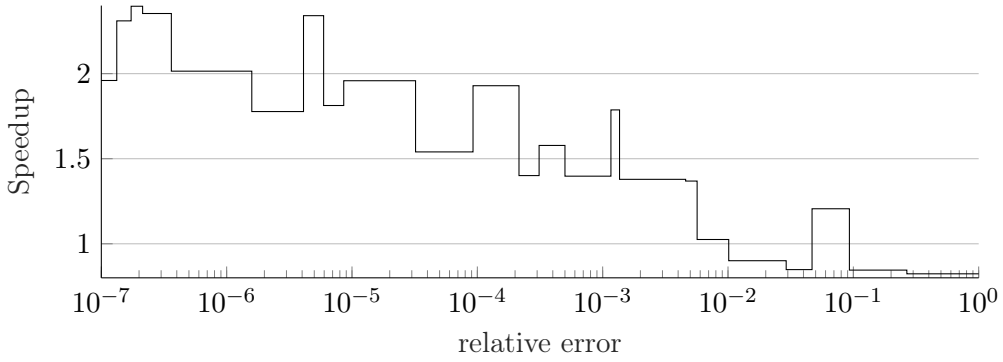

Figure 18: `wt_nx21` model: Speedup of GNSF-IRK w.r.t. the standard IRK, using an approximation of the efficient set similar to Figure 15.

Figure 18 shows that for the `wt_nx21` model it makes sense to use GNSF-IRK if accuracies higher than 10^{-2} are required. However, as this system is a more detailed model of a wind turbine, compared to the `wt_nx13` model, the model mismatch is supposed to be smaller, which makes longer sampling times and high accuracy integrators more attractive for MPC. Thus, for this model, one can expect a speedup greater than 50% by using GNSF-IRK.

Figure 19 shows that the structure within the `wt_nx6` model is really well suited for GNSF-IRK. Given any accuracy, one can obtain a positive speedup by using the GNSF-IRK scheme. However, this speedup is always in the range of 5% to 34%. The reason for this relatively small speedup is that the spline evaluations used in the model are very costly, which we already saw in the previous sections of this chapter.

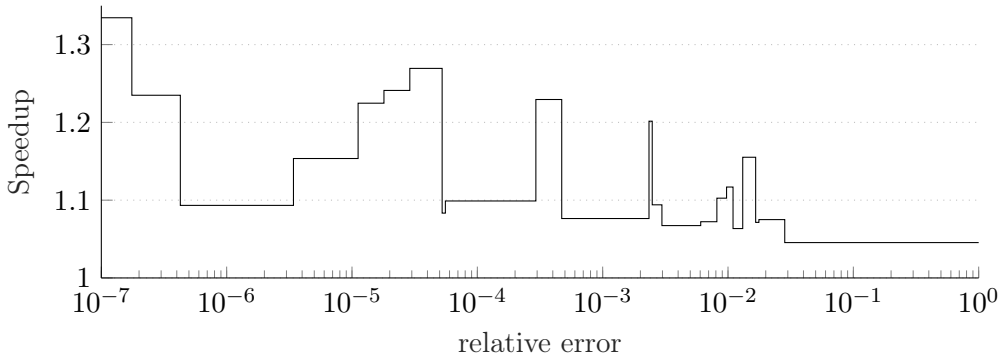

Figure 19: `wt_nx6` model: Speedup of GNSF-IRK w.r.t. the standard IRK, using an approximation of the efficient set similar to Figure 15.

5.6 Sparsity of Bold Matrices

In this section, we want to look at the bold matrices $\mathbf{Y}_\mathrm{x}$, $\mathbf{Y}_\mathrm{u}$, $\mathbf{Y}_\mathbf{v}$, $\mathbf{K}_\mathrm{x}$, $\mathbf{K}_\mathrm{u}$, $\mathbf{K}_\mathbf{v}$, $\mathbf{Z}_\mathrm{x}$, $\mathbf{Z}_\mathrm{u}$, $\mathbf{Z}_\mathbf{v}$ with a focus on their sparsity pattern, i.e. the positions of nonzero matrix entries. Let us refer to these matrices as *GNSF-IRK matrices*, because they are the ones that are actually used within GNSF-IRK. We saw in Section 5.2 that the deployment of GNSF-IRK comes with the computational burden of multiplying with these matrices. Within GNSF-IRK, these operations are often more costly compared to the linear system solutions. Thus, it makes sense to investigate possibilities to tune these operations.

In Figure 20, the sparsity patterns of these matrices are plotted for the `wt_nx6` model using Gauss-Legendre tableaus of different sizes s. In addition to the sparsity patterns, gray lines are plotted to show the block structure of the matrices. Since the `wt_nx6` model is an ODE, the matrices $\mathbf{Z}_\mathrm{x}, \mathbf{Z}_\mathrm{u}, \mathbf{Z}_\mathbf{v}$ are empty and not listed.

We observe that the matrices with index x and u consist of blocks that always have the same sparsity pattern and are vertically concatenated. The structure of these blocks originates from the structure inherent within the dynamic model.

The matrices with index $\mathbf{v}$ grow in both directions with increasing s. Regarding the matrix $\mathbf{Y}_\mathbf{v}$ for this model, we note that all blocks have the same simple structure. On the other hand, we see that this is not the case for the matrix $\mathbf{K}_\mathbf{v}$. However, all blocks of $\mathbf{K}_\mathbf{v}$ only have nonzero elements at positions where the initial block, i.e. $\mathbf{K}_\mathbf{v}$ for $s = 1$, has a nonzero entry.

The same properties also hold for the matrices corresponding to the `inv_pend` model, which are shown in the Figures 21 and 22. Namely, the matrices with index x and u consist of blocks that have the exact same sparsity pattern as their initial block, i.e. the corresponding matrix for $s = 1$. The matrices with index $\mathbf{v}$ consist of blocks that have the same sparsity pattern as their initial block, but there are some additional

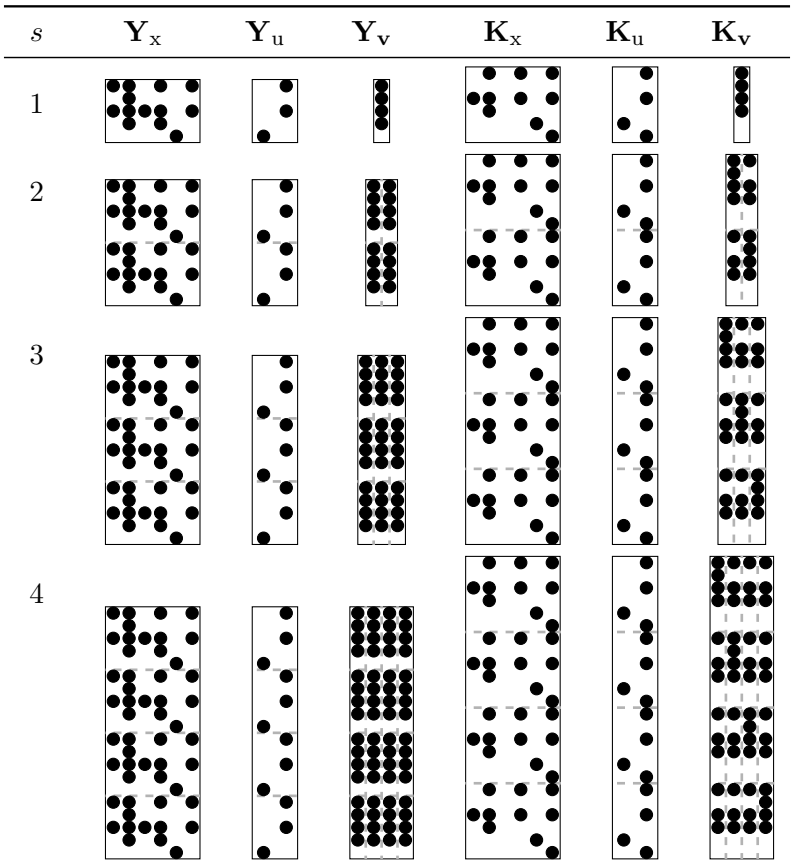

Figure 20: `wt_nx6` model: Sparsity patterns of the GNSF-IRK matrices.

zero entries in the subsequent blocks.
It could be possible to derive the described properties for the GNSF-IRK matrices formally using their definitions. At least with the regarded test problems, we were not able to find a counter example.

We note that the bold matrices for the `wt_nx6` model are not very sparse and using BLASFEO routines to multiply them is computationally efficient.
In order to figure out if BLASFEO routines are generally an efficient way to perform these computations, we want to get an overview on the sparsity of the bold matrices for the other test models, without looking at the patterns.
For this purpose, we calculate the percentages of nonzero entries within the GNSF-IRK matrices, which are given in Table 15 and Table 16.
Since the GNSF-IRK matrices with index x and u have the same sparsity ratio for all s tested, they are listed together in Table 15. We have seen that the ratio of nonzero elements within the GNSF-IRK matrices with index $\mathbf{v}$ generally varies with the number of stages s, when looking at the sparsity patterns in Figures 20, 21 and 22.

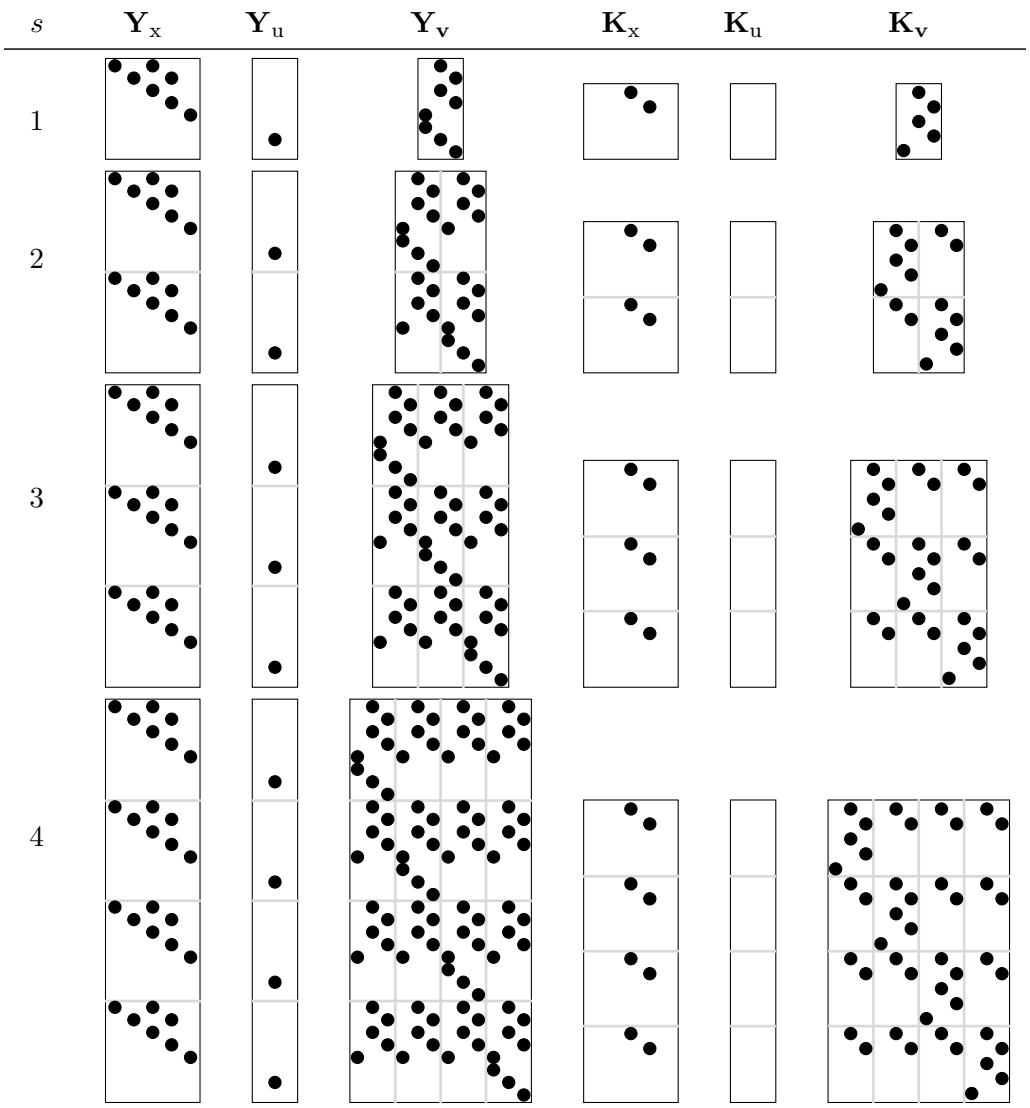

Figure 21: `inv_pend` model: Sparsity patterns of the GNSF-IRK matrices.

However, the percentage of nonzero entries within the GNSF-IRK matrices using the `wt_nx13` and `wt_nx21` model is constant, i.e. all blocks in these matrices have exactly the same structure.
Comparing the three wind turbine models, one observes that the ones with more states result in GNSF-IRK matrices that are more sparse. For the `wt_nx13` model, the GNSF-IRK matrices consist of 19 to 24% of nonzero entries. Using the `wt_nx21` model, only around 10% of the entries in the GNSF-IRK matrices are nonzero.

We want to briefly discuss the estimated difference in computation time, when using sparse linear algebra routines instead of BLASFEO. Modern CPU architectures

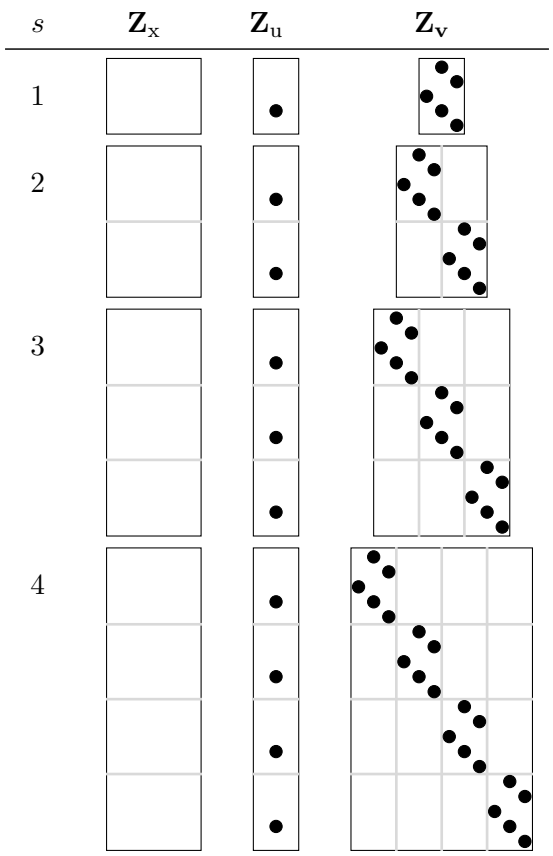

Figure 22: `inv_pend` model: Sparsity patterns of the GNSF-IRK matrices, Z matrices.

have vector floating point units. Typically, these units can operate on four doubles with a single instruction, giving an approximately 4 times speedup over the scalar code in sparse computations. Efficient implementations that use vectorization are available within the BLASFEO HP (high performance) version. Additionally, within BLASFEOs panel-major matrix format, matrix elements are stored in the same order as they are typically accessed by the BLASFEO routines such that the computational kernel of the CPU can be continuously fed [3]. Both points are not true for sparse linear algebra routines. In combination, one can expect that the usage of sparse linear algebra routines can only outperform BLASFEO if the percentage of nonzero entries is lower than around 10%.

Regarding again Table 15 and Table 16 with respect to the previous discussion, we can state that for the regarded test problems the deployment of efficient BLASFEO routines is indeed a good choice given the alternative of sparse linear algebra. Only for the largest wind turbine model, `wt_nx21`, the percentage of nonzero entries within the GNSF-IRK matrices is at a level for which sparse linear algebra routines could potentially compete with BLASFEO.

The discussion in this section can be summarized as follows. For the regarded test problems, the usage of high performing linear algebra routines, such as the ones in BLASFEO, is a suitable choice for the operations of interest. If GNSF-IRK is used with larger models, such that the GNSF-IRK matrices are very sparse, it could make sense to use sparse linear algebra routines instead.
However, this thesis already proposed two other approaches to reduce the computational burden of operations that involve the GNSF-IRK matrices, which we will briefly recall.
First, Section 3.4.4 suggested to use `CasADi` code generation more extensively, which would not only exploit the sparsity within the GNSF-IRK matrices, but also the sparsity pattern within the Jacobian. Moreover, this approach could also yield cheaper derivative evaluations, since the chain rule, e.g. in (3.29), could be carried out using advanced AD algorithms in `CasADi`.
Second, Section 3.4.5 suggests to extend the automatic transcription method to detect also multi-stage GNSF structures. It should be further investigated how frequently these structures occur within NMPC models. Hereby, the focus should be on large dynamic system models that result in very sparse GNSF-IRK matrices, if the proposed single-stage GNSF-IRK algorithm is deployed.

	$\mathbf{Y}_\mathrm{x}$	$\mathbf{K}_\mathrm{x}$	$\mathbf{Z}_\mathrm{x}$	$\mathbf{Y}_\mathrm{u}$	$\mathbf{K}_\mathrm{u}$	$\mathbf{Z}_\mathrm{u}$
`wt_nx6`	43	28	-	30	33	-
`wt_nx13`	22	24	-	19	23	-
`wt_nx21`	10	11	-	7.4	12	-
`inv_pend`	18	8	0	12	0	20
`inv_pend` naive	15	8.3	0	38	33	40

Table 15: Percentage of nonzero entries within the GNSF-IRK matrices with index x or u for all test problems.

	$\mathbf{Y}_\mathbf{v}$		$\mathbf{K}_\mathbf{v}$		$\mathbf{Z}_\mathbf{v}$	
	min	max	min	max	min	max
`wt_nx6`	80	80	53	67	-	-
`wt_nx13`	20	20	20	20	-	-
`wt_nx21`	10	10	10	10	-	-
`inv_pend`	23	33	17	33	5.6	33
`inv_pend` naive	19	24	15	29	2.7	16

Table 16: Percentage of nonzero entries within the GNSF-IRK matrices with index **v** for all test problems. The maximum and minimum of this ratio are taken regarding $s = 1, \ldots, 6$.

6 Conclusions and Outlook

This thesis proposed a new structure exploiting IRK scheme suitable for Model Predictive Control. It further aimed at giving an overview on the concept of integrators for embedded optimization software. The work mainly focused on IRK methods with sensitivity propagation for index-1 DAE systems, which are suitable for typical NMPC applications. The state-of-the-art implementation of an IRK scheme was outlined and discussed. Previous work on structure exploitation of the Newton matrix within an IRK scheme was presented and compared. The approaches to exploit linear dependencies within the underlying dynamic system were investigated with the intention to combine and extend them. The result was a general dynamic system structure that can also handle index-1 DAEs and is called GNSF. An efficient IRK scheme (GNSF-IRK) that exploits the proposed structured dynamic system and that can also propagate first order sensitivities was derived.
The structure exploiting IRK scheme has been implemented in `C` within the `acados` framework using BLASFEO to perform linear algebra operations efficiently. In order to make this implementation conveniently usable, an automatic transcription method has been derived, which is able to transcribe most index-1 DAE systems into the GNSF form, such that the proposed GNSF-IRK scheme can be deployed efficiently. Both algorithms were successfully tested with several models, some of which were kindly supplied by our industrial partners within the eco4wind project, the goal of which the goal is to develop NMPC controllers for wind turbine generators.
The properties of the proposed GNSF-IRK scheme were discussed in detail, including the evolution and distribution of computation time and how the error compares to the one of the standard IRK implementation.
It was concluded, that an optimal choice of options for the IRK algorithm is depending on whether the structure exploiting GNSF-IRK or a standard IRK implementation is used. We found that an optimal choice of options for GNSF-IRK uses a rather high integration order with a relatively low number of steps compared to the standard implementation.
The numerical experiments showed that the usage of GNSF-IRK with an appropriately GNSF reformulated model typically results in a speedup, which is however very depending on the model and the desired accuracy. The speedup factors we found in realistic applications vary between 1 to 3.8.

Finally, possible extensions of the proposed algorithms and improvements in their implementations have been critically discussed and classified.

The potential of extending the proposed algorithms to a multi-stage GNSF formulation should be investigated further. An approach to use `CasADi` code generation more extensively within the implementation should be benchmarked against the current implementation. Moreover, the extension of the proposed GNSF-IRK scheme to optionally propagate second order sensitivity information and the implementation of a lifted Newton version of the scheme was left out of the scope. These aspects and their benchmarking within NMPC schemes are interesting directions for further research.

Appendix – Model Equations corresponding to the `wt_nx21` Model

The appendix only consists of the model equations that describe the dynamics of q within the `wt_nx21` model, see Section 5.1.3.

$$0 = (-\ddot{q}_9 - \ddot{q}_7 - \ddot{q}_5)\, p_{20} \sin\vartheta + (\ddot{q}_8 + \ddot{q}_6 + \dot{q}_4)\, p_{19} \cos\vartheta + \ddot{q}_1 p_9 + \ddot{q}_1 p_8 + \ddot{q}_1 p_{29} + q_1 p_{28} + \dot{q}_1 p_{27} + 3\ddot{q}_1 p_{22} - F_{\text{thrust}} \tag{6.1a}$$

$$0 = \frac{1}{2}\Big(\, 2p_1 T_{\text{rot}} + \Big(\Big(\sqrt{3}\ddot{q}_8 + 2\dot{q}_3\dot{q}_8 - \sqrt{3}\ddot{q}_6 + 2\dot{q}_3\dot{q}_6 - 4\dot{q}_3\dot{q}_4\Big)\sin q_3 + \Big(-\ddot{q}_8 + 2\sqrt{3}\dot{q}_3\dot{q}_8 - \ddot{q}_6 - 2\sqrt{3}\dot{q}_3\dot{q}_6 + 2\ddot{q}_4\Big)\cos q_3\Big)\, p_{19}\sin\vartheta + \dots \tag{6.1b}$$
$$\Big(\Big(\sqrt{3}\ddot{q}_9 + 2\dot{q}_3\dot{q}_9 - \sqrt{3}\ddot{q}_7 + 2\dot{q}_3\dot{q}_7 - 4\dot{q}_3\dot{q}_5\Big)\sin q_3 + \Big(-\ddot{q}_9 + 2\sqrt{3}\dot{q}_3\dot{q}_9 - \ddot{q}_7 - 2\sqrt{3}\dot{q}_3\dot{q}_7 + 2\ddot{q}_5\Big)\cos q_3\Big)\, p_{20}\cos\vartheta + 2\ddot{q}_2 p_9 + 2\ddot{q}_2 p_8 + 2\ddot{q}_2 p_{30} + 2q_2 p_{28} + 2\dot{q}_2 p_{27} + 6\ddot{q}_2 p_{22}\Big)$$

$$0 = -T_{\text{rot}} + (\ddot{q}_8 + \ddot{q}_6 + \ddot{q}_4)\, p_{17}\sin\vartheta + (-\ddot{q}_9 - \ddot{q}_7 - \ddot{q}_5)\, p_{18}\cos\vartheta + (q_3 - q_{10})\, p_7 + \ddot{q}_3 p_6 + 3\ddot{q}_3 p_{21} + (\dot{q}_3 - \dot{q}_{10})\, p_2 \tag{6.1c}$$

$$0 = -\frac{p_{26}\sin\vartheta T_{\text{rot}} + p_{19}\left((-3\sin q_3 p_3 - 3\ddot{q}_2\cos q_3)\sin\vartheta - 3\ddot{q}_1\cos\vartheta\right) - 3\ddot{q}_3 p_{17}\sin\vartheta + F_{\text{thrust}} p_{24}\cos\vartheta - 3\ddot{q}_4 p_{15} - 3q_4 p_{13} - 3\dot{q}_4 p_{11}}{3} \tag{6.1d}$$

$$0 = \frac{p_{25}\cos\vartheta T_{\text{rot}} + p_{20}\left((3\sin q_3 p_3 + 3\ddot{q}_2\cos q_3)\cos\vartheta - 3\ddot{q}_1\sin\vartheta\right) - F_{\text{thrust}} p_{23}\sin\vartheta - 3\ddot{q}_3 p_{18}\cos\vartheta + 3\ddot{q}_5 p_{16} + 3q_5 p_{14} + 3\dot{q}_5 p_{12}}{3} \tag{6.1e}$$

$$0 = -\frac{2p_{26}\sin\vartheta T_{\text{rot}} + p_{19}\left(\left(\left(3\sin q_3 - 3^{\frac{3}{2}}\cos q_3\right)p_3 + 3^{\frac{3}{2}}\ddot{q}_2\sin q_3 + 3\ddot{q}_2\cos q_3\right)\sin\vartheta - 6\ddot{q}_1\cos\vartheta\right) - 6\ddot{q}_3 p_{17}\sin\vartheta + 2F_{\text{thrust}} p_{24}\cos\vartheta - 6\ddot{q}_6 p_{15} - 6q_6 p_{13} - 6\dot{q}_6 p_{11}}{6} \tag{6.1f}$$

$$0 = \frac{2p_{25}\cos\vartheta T_{\text{rot}} + p_{20}\left(\left(\left(3^{\frac{3}{2}}\cos q_3 - 3\sin q_3\right)p_3 - 3^{\frac{3}{2}}\ddot{q}_2\sin q_3 - 3\ddot{q}_2\cos q_3\right)\cos\vartheta - 6\ddot{q}_1\sin\vartheta\right) - 2F_{\text{thrust}} p_{23}\sin\vartheta - 6\ddot{q}_3 p_{18}\cos\vartheta + 6\ddot{q}_7 p_{16} + 6q_7 p_{14} + 6\dot{q}_7 p_{12}}{6} \tag{6.1g}$$

$$0 = -\frac{2p_{26}\sin\vartheta T_{\text{rot}} + p_{19}\left(\left(\left(3\sin q_3 + 3^{\frac{3}{2}}\cos q_3\right)p_3 - 3^{\frac{3}{2}}\ddot{q}_2\sin q_3 + 3\ddot{q}_2\cos q_3\right)\sin\vartheta - 6\ddot{q}_1\cos\vartheta\right) - 6\ddot{q}_3 p_{17}\sin\vartheta + 2F_{\text{thrust}} p_{24}\cos\vartheta - 6\ddot{q}_8 p_{15} - 6q_8 p_{13} - 6\dot{q}_8 p_{11}}{6} \tag{6.1h}$$

$$0 = \frac{2p_{25}\cos\vartheta T_{\text{rot}} + p_{20}\left(\left(\left(-3\sin q_3 - 3^{\frac{3}{2}}\cos q_3\right)p_3 + 3^{\frac{3}{2}}\ddot{q}_2\sin q_3 - 3\ddot{q}_2\cos q_3\right)\cos\vartheta - 6\ddot{q}_1\sin\vartheta\right) - 2F_{\text{thrust}} p_{23}\sin\vartheta - 6\ddot{q}_3 p_{18}\cos\vartheta + 6\ddot{q}_9 p_{16} + 6q_9 p_{14} + 6\dot{q}_9 p_{12}}{6} \tag{6.1i}$$

$$0 = p_{10}\, T_{\text{gen}} + (q_{10} - q_3)\, p_7 + {p_{10}}^2\,(\ddot{q}_{10} p_5 + \ddot{q}_{10} p_4) + (\dot{q}_{10} - \dot{q}_3)\, p_2 \tag{6.1j}$$

Bibliography

[1] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, R. Quirynen, and M. Diehl, "Towards a modular software package for embedded optimization," 2018.

[2] acados, "acados." `http://acados.org`, 2017. [Online; accessed 30-July-2018].

[3] 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.

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

[5] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 26, no. 6, pp. 789–814, 2000.

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

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

[8] E. Zafiriou, "Robust model predictive Control of processes with hard constraints," *Computers & Chemical Engineering*, vol. 14, no. 4–5, pp. 359–371, 1990.

[9] R. Quirynen, M. Vukov, and M. Diehl, "Multiple shooting in a microsecond," in *Contributions in Mathematical and Computational Sciences*, pp. 183–201, Springer, 2015.

[10] C. Kirches, L. Wirsching, S. Sager, and H. G. Bock, "Efficient numerics for nonlinear model predictive control," in *Recent Advances in Optimization and its Applications in Engineering*, pp. 339–357, Springer, 2010.

[11] E. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. New York: McGraw-Hill, 1955.

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

[13] J. Andersson, J. Åkesson, and M. Diehl, "Dynamic optimization with CasADi," 2012.

[14] M. Diehl and S. Gros, *Numerical Optimal Control - script draft.* 2017. (Available online: `https://www.syscop.de/files/2017ss/NOC/script/book-NOCSE.pdf`).

[15] B. Saerens, M. Diehl, and E. Van den Bulck, *Automotive Model Predicitive Control: Models, Methods and Applications*, vol. 402/2010 of *Lecture Notes in Control and Information Sciences*, ch. Optimal Control Using Pontryagin's Maximum Principle and Dynamic Programming, pp. 119–138. Berlin / Heidelberg: Springer, 2010.

[16] D. P. Word, J. Kang, J. Akesson, and C. D. Laird, "Efficient parallel solution of large-scale nonlinear dynamic optimization problems," *Computational Optimization and Applications*, vol. 59, no. 3, pp. 667–688, 2014.

[17] 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.

[18] H. J. Ferreau, B. Houska, T. Kraus, and M. Diehl, "Numerical methods for embedded optimisation and their implementation within the ACADO toolkit," in *In Proceedings of the 7th Conference – Computer Methods and Systems* (W. M. R. Tadeusiewicz, A. Ligeza and M. Szymkat, eds.), (Krakow, Poland), pp. 13–29, Oprogramowanie Naukowo-Techniczne, November 2009.

[19] R. Quirynen, "Automatic code generation of Implicit Runge-Kutta integrators with continuous output for fast embedded optimization," Master's thesis, KU Leuven, 2012.

[20] L. T. Biegler, *Nonlinear Programming.* MOS-SIAM Series on Optimization, SIAM, 2010.

[21] J. Nocedal and S. J. Wright, *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering, Springer, 2 ed., 2006.

[22] S. Bartels, *Numerik* 3×9*. Drei Themengebiete in jeweils neun kurzen Kapiteln.* Heidelberg: Springer Spektrum, 2016.

[23] W. Murray, "Newton-type methods." `https://web.stanford.edu/class/cme334/docs/newton-type-methods.pdf`, 2010.

[24] "ACADO Toolkit." `http://www.acadotoolkit.org`, 2009–2016.

[25] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit – an open source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[26] R. Quirynen, S. Gros, and M. Diehl, "Fast auto generated ACADO integrators and application to MHE with multi-rate measurements," pp. 3077–3082, 2013.

[27] 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.

[28] "CasADi." `http://casadi.org`, 2013. [Online; accessed 26-October-2018].

[29] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your Jacobian? Graph coloring for computing derivatives," *SIAM Review*, vol. 47, pp. 629–705, 2005.

[30] S. Gros, M. Zanon, M. Vukov, and M. Diehl, "Nonlinear MPC and MHE for Mechanical Multi-Body Systems with Application to Fast Tethered Airplanes," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, pp. 86–93, 2012.

[31] 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.

[32] S. Gros and M. Diehl, "Numerical optimal control with differential algebraic equations - lecture notes." 2016.

[33] G. Fábián, D. Van Beek, and J. Rooda, "Index reduction and discontinuity handling using substitute equations," vol. 7, pp. 173–187, 06 2001.

[34] J. Albersmeyer, *Adjoint-based algorithms and numerical methods for sensitivity generation and optimization of large scale dynamic systems*. PhD thesis, Ruprecht-Karls-Universitität Heidelberg, 2010.

[35] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*. Springer Series in Computational Mathematics, Berlin: Springer, 2nd ed., 1993.

[36] G. Dahlquist, "Convergence and stability in numerical integration of ordinary differential equations," *Math. Scand.*, vol. 4, pp. 33–53, 1956.

[37] E. Eich-Soelner and C. Führer, *Numerical Methods in Multibody Dynamics*. European Consortium for Mathematics in Industry (ECMI series), Teubner, 1998.

[38] R. Quirynen, S. Gros, and M. Diehl, "Inexact Newton based lifted implicit integrators for fast nonlinear MPC," pp. 32–38, 2015.

[39] R. Quirynen, M. Vukov, and M. Diehl, "Auto generation of implicit integrators for embedded NMPC with microsecond sampling times," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference* (M. Lazar and F. Allgöwer, eds.), pp. 175–180, 2012.

[40] R. Quirynen, B. Houska, and M. Diehl, "Symmetric hessian propagation for lifted collocation integrators in direct optimal control.," pp. 1117–1123, 2016.

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

[42] T. A. Bickart, "An efficient solution process for implicit Runge-Kutta methods," *SIAM Journal on Numerical Analysis*, vol. 14, no. 6, pp. 1022–1027, 1977.

[43] S. González-Pinto, J. I. Montijano, and L. Rández, "Iterative schemes for three-stage implicit Runge-Kutta methods," *Appl. Numer. Math.*, vol. 17, pp. 363–382, Aug. 1995.

[44] S. González-Pinto, S. Pérez-Rodríguez, and J. I. Montijano, "Implementation of high-order implicit Runge-Kutta methods," *Computers & Mathematics with Applications*, vol. 41, no. 7-8, pp. 1009–1024, 2001.

[45] G. Cooper and R. Vignesvaran, "Some schemes for the implementation of implicit Runge-Kutta methods," *Journal of Computational and Applied Mathematics*, vol. 45, no. 1–2, pp. 213–225, 1993.

[46] R. Quirynen, S. Gros, and M. Diehl, "Efficient NMPC for nonlinear models with linear subsystems," pp. 5101–5106, 2013.

[47] 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," 2017.

[48] S. Gros and M. Diehl, "NMPC based on Huber penalty functions to handle large deviations of quadrature states," pp. 3159–3164, 2013.

[49] "SWIG." `http://swig.org/`, 2014. [Online; accessed 26-October-2018].

[50] C. Büskens and M. Knauer, "Higher Order Real-Time Approximations in Optimal Control of Multibody-Systems for Industrial Robots," *Multibody System Dynamics*, vol. 15, no. 1, pp. 85–106, 2006.

[51] "acados github branch including the numerical experiments.." `https://github.com/FreyJo/acados/tree/numerical_experiments`. [Online; accessed 5-November-2018].

[52] S. Gros, R. Quirynen, and M. Diehl, "An Improved Real-time NMPC Scheme for Wind Turbine Control using Spline-Interpolated Aerodynamic Coefficients," 2014.

[53] 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 *Accepted for the 2018 European Control Conference ECC*, 2018.

[54] G. Eichfelder, *Vector Optimization in Medical Engineering*, pp. 181–215. New York, NY: Springer New York, 2014.

[55] "Pareto Front for Two Objectives - Matlab Documentation." `https://de.mathworks.com/help/gads/pareto-front-for-two-objectives.html`. [Online; accessed 15-October-2018].

[56] A. Zanelli, G. Horn, G. Frison, and M. Diehl, "Nonlinear model predictive control of a human-sized quadrotor," 2018.

[57] S. Gros, M. Vukov, and M. Diehl, "A real-time MHE and NMPC scheme for wind turbine control," 2013.