

KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculteit
Ingenieurswetenschappen

# Automatic code generation of Implicit Runge-Kutta integrators with continuous output for fast embedded optimization

Rien Quirynen

Thesis voorgedragen tot het
behalen van de graad van Master
in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken

**Promotor:**
Prof. Dr. Ir. Moritz Diehl

**Assessoren:**
Prof. Dr. Ir. Stefan Vandewalle
Prof. Dr. Ir. Paul Dierckx

**Begeleiders:**
Ir. Milan Vukov
Dr. Hans Joachim Ferreau

Academiejaar 2011 – 2012

© Copyright K.U.Leuven

Without written permission of the thesis supervisor and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email `info@cs.kuleuven.be`.

A written permission of the thesis supervisor is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail `info@cs.kuleuven.be`.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

# Preface

I would like to thank my promotor and assistants for this project and for their motivated support during the last year. Especially, I want to thank my promotor Moritz Diehl for his enthusiastic way of providing new ideas and motivating people. He surely ensured that I was able to achieve as much as possible with my thesis project. I also want to thank my daily supervisor Milan Vukov in particular for always making time for a meeting and helping me out. He always has the critical view, necessary to make sure everything is correct. I would like to thank Hans Joachim Ferreau and Boris Houska for inspiring discussions and for the ACADO Toolkit. I want to thank Mario Zanon and Sébastien Gros for their cooperation by providing multiple test problems. I also want to thank the whole research team of Moritz Diehl for the pleasant work atmosphere. And eventually, I want to thank the jury for reading the text.

*Rien Quirynen*

# Contents

**Preface** i

**Abstract** iv

**List of Figures** v

**List of Tables** vii

**List of Abbreviations and Symbols** ix

**1 Introduction** 1
- 1.1 Motivation . . . 1
- 1.2 Real-Time Optimization . . . 2
- 1.3 The ACADO Toolkit . . . 7
- 1.4 Online Algorithms . . . 8
- 1.5 Code Generation . . . 9
- 1.6 Contents . . . 12

**2 Integration methods** 13
- 2.1 Runge-Kutta Methods . . . 13
- 2.2 Collocation Methods . . . 17
- 2.3 Stiff systems of equations . . . 22
- 2.4 Solving the nonlinear system . . . 26
- 2.5 Initialization of the Variables . . . 31
- 2.6 Conclusion . . . 33

**3 Derivative Generation** 34
- 3.1 Implicit Function Theorem (IFT) . . . 35
- 3.2 Internal Numerical Differentiation (IND) . . . 37
- 3.3 Variational Differential Equations (VDE) . . . 40
- 3.4 Adjoint sensitivity generation . . . 41
- 3.5 Conclusion . . . 41

**4 Simulation of Differential-Algebraic Equations** 43
- 4.1 Differential-Algebraic Equations . . . 43
- 4.2 Derivative Generation . . . 48
- 4.3 Conclusion . . . 51

**5 Continuous Output** 52
- 5.1 Motivation . . . 52

5.2 Implementation . . . 53
5.3 Continuous extension of methods . . . 55
5.4 Conclusion . . . 55

**6 Numerical Experiments** **56**
6.1 Test Problems with ODE Systems . . . 56
6.2 Test Problems with DAE Systems . . . 59
6.3 Choice of the Standard Parameters . . . 62
6.4 Efficient Derivative Generation . . . 64
6.5 Initialization of the Variables . . . 67
6.6 Solving the Algebraic equations . . . 70
6.7 Performance of the integrators . . . 72
6.8 Impact on Embedded Optimization . . . 78

**7 Conclusion** **80**

**A NMPC paper** **83**

**B Poster** **90**

**C Software implementation** **92**
C.1 IntegratorExport . . . 93
C.2 ExportLinearSolver . . . 95
C.3 SIMexport . . . 96

**D Extra numerical results** **100**
D.1 Choice of the Standard Parameters . . . 100
D.2 Efficient Derivative Generation . . . 101
D.3 Initialization of Subsequent Steps . . . 103
D.4 Solving the Algebraic equations . . . 104

**Bibliography** **105**

# Abstract

Nonlinear Model Predictive Control (NMPC) and Moving Horizon Estimation (MHE) have become quite popular approaches to achieve optimal control and state estimation. The use of Nonlinear MPC or MHE for real-time control of mechatronic systems with fast dynamics is still a major challenge. A combination of efficient online algorithms and code optimization is needed. The principle of automatic generation of optimized code has spread in the world of embedded optimization. Examples are CVXGEN for real-time convex optimization, the Multi-Parametric Toolbox (MPT) for explicit MPC and the code generation tool of the ACADO Toolkit. The latter implements the Real-Time Iteration (RTI) scheme. Quite recently there has been made some important progress with the code generation tool of ACADO. This thesis project also participated in the growth of this tool and can be characterized by three major aspects.

The RTI algorithm is based on a shooting discretization of the OCP. The integration of the system with sensitivity generation therefore forms a major computational step. Until now, this was tackled using code generation of the Explicit Runge-Kutta (ERK) method of order 4. The first aspect of this thesis project is to show that auto generation of Implicit Runge-Kutta (IRK) methods with sensitivity generation can also be implemented very efficiently. Implicit integrators improve the support for stiff systems of equations and also allow one to handle systems of Differential Algebraic Equations (DAE) instead of only Ordinary Differential Equations (ODE). The second major aspect is therefore to efficiently extend the methods to DAE systems of index 1.

Collocation methods, which are a special case of IRK methods that can provide a continuous approximation of the solution, are quite promising for MHE with high frequency measurements. They allow the integration step size to be larger than imposed by the measurement frequency, while still being able to use all the high frequency data without any loss of information. The continuous output of these methods could enable many other applications. To translate this into an optional feature of the methods that can efficiently support all of these applications is the third aspect.

The final result of this project besides this text is the well documented open source code, made part of the ACADO code generation tool.

# List of Figures


1.1 Estimation-based Feedback Control: This figure illustrates the interaction between the system and the estimation (e.g. MHE) and control algorithm (e.g. MPC). . . . . 2
1.2 Illustration of the estimation horizon (MHE) and the prediction horizon (NMPC), assuming $k$ is the current time. . . . . 4
1.3 Illustration of one shooting interval from time point $k$ to $k+1$, assuming multiple fixed integration steps are taken. . . . . 6
1.4 This figure illustrates the principle of code generation to obtain a custom solver for a certain problem family [45]. . . . . 9

2.1 Structure of the matrix $A$ for different families of RK methods [41]. . . 16
2.2 An example of a comparison between the relative error at the quadrature points and in the middle of these points with respect to the step size $h$ for the $4^{\text{th}}$ order Gauss method. . . . . 21
2.3 The relative stability diagrams for the 3 Gauss-Legendre methods. . . . 25

5.1 The red dotted line represents a certain constraint which gets violated in between two discretization points, denoted by the vertical bars. . . . . 53

6.1 The eigenvalues of the Jacobian of the crane model for its initial values. 58
6.2 The eigenvalues of the Jacobian of the kite ODE model for its initial values. 59
6.3 A simple planar pendulum. . . . . 60
6.4 Crane model: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method. . . . . 63
6.5 Elastic kite model: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities. . . . 66
6.6 Crane model: 2 work-precision diagrams for the order 2 and order 4 IRKC with IFT-R method to study the impact of performing 0, 1 or 2 extra Newton iterations during the first integration step. . . . . 68
6.7 Crane model: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables. . . . . 70

6.8 Pendulum model: The work-precision diagram for the IRKC IFT-R method to study the impact of the number of Newton iterations for the separate system of algebraic equations. . . . . 71
6.9 The work-precision diagram comparing different methods for the integration of the kite DAE model with extra outputs at 1kHz. It presents the total computation time with respect to the mean relative error of the outputs. . . . . 78

C.1 Inheritance diagram to highlight the most important classes, implemented in the context of this thesis project. . . . . 93

D.1 Van der Pol oscillator: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method. . . . . 100
D.2 Elastic kite model: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method. . . . . 101
D.3 Van der Pol oscillator: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities. . . . 102
D.4 Crane model: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities. . . . . 102
D.5 Van der Pol oscillator: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables. . . . . 103
D.6 Elastic kite model: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables. . . . . 103
D.7 Kite DAE model: The work-precision diagram for the IRKC IFT-R method to study the impact of the number of Newton iterations for the separate system of algebraic equations. . . . . 104

# List of Tables


1.1 Worst-case timing results of the auto-generated real-time iteration algorithm applied to the kite carousel model [23]. . . . . 11

6.1 Average computation time per integration step for overhead crane example. 67
6.2 Composition of the average computation time per integration step for the $4^{\text{th}}$ order Gauss method on crane model. . . . . 67
6.3 Integration of the Van der Pol oscillator over 0.1s. . . . . 73
6.4 Integration of the crane model over 0.1s. . . . . 73
6.5 Integration of the kite ODE model over 0.1s. . . . . 74
6.6 Integration of the pendulum over 1s. . . . . 75
6.7 Integration of the kite DAE model over 1s. . . . . 75
6.8 Order of accuracy and of the continuous output for the 1-, 2- and 3-stage Gauss-Legendre and Radau IIA methods. . . . . 76
6.9 Integration of the kite DAE model over 1s with extra outputs at 1kHz. . 77
6.10 Average computation times for NMPC of an overhead crane. . . . . 79

# List of Abbreviations and Symbols

## Abbreviations

| | |
|---|---|
| RTI | Real-Time Iteration |
| OCP | Optimal Control Problem |
| MS | Multiple Shooting |
| NLP | Nonlinear Program |
| LP | Linear Program |
| QP | Quadratic Program |
| MPC | Model Predictive Control |
| NMPC | Nonlinear MPC |
| MHE | Moving Horizon Estimation |
| KKT | Karush-Kuhn-Tucker |
| ODE | Ordinary Differential Equation |
| DAE | Differential Algebraic Equation |
| IVP | Initial Value Problem |
| BDF | Backward Differentiation Formula |
| RK | Runge-Kutta |
| ERKn | Explicit Runge-Kutta of order n |
| IRKn | Implicit Runge-Kutta of order n |
| IRKCn | Implicit Runge-Kutta of order n with Continuous output |
| DIRK | Diagonal Implicit Runge-Kutta |
| SDIRK | Singly Diagonal Implicit Runge-Kutta |
| ESDIRK | Explicit Singly Diagonal Implicit Runge-Kutta |
| IND | Internal Numerical Differentiation |
| END | External Numerical Differentiation |
| AD | Algorithmic Differentiation |
| IND-AD | Internal Numerical Differentiation using Algorithmic Differentiation |
| IFT | Implicit Function Theorem |
| IFT-R | Implicit Function Theorem and Reuse of Jacobian |
| FD | Finite Differences |
| VDE | Variational Differential Equations |
| GPS | Global Positioning System |
| IMU | Inertial Measurement Unit |

## Symbols

| | |
|---|---|
| $\mathbb{0}$ | The zero matrix |
| $\mathbb{1}$ | The identity matrix |
| $\|\cdot\|_M$ | Norm weighted by a symmetric, positive semi-definite matrix $M$ |
| $i, j$ | Subscript indices denoting components |
| $f$ | The function in the right-hand side of an ODE |
| $g$ | The function defining the algebraic equations |
| $x$ | The differential states |
| $z$ | The algebraic states |
| $u$ | Control inputs |
| $p$ | Model parameters |
| $w$ | The disturbances |
| $t$ | Current time |
| $N_x$ | Number of differential states |
| $N_z$ | Number of algebraic states |
| $N_u$ | Number of control inputs |
| $N_p$ | Number of model parameters |
| $h$ | Integration step size |
| $s$ | Number of stages of a RK method |
| $A$ | The internal coefficients of a RK method |
| $b$ | The weights of a RK method |
| $c$ | The nodes of a RK method |
| $X$ | Stage values for the differential states |
| $Z$ | Stage values for the algebraic states |
| $k$ | The variables of a RK method for an ODE |
| $K$ | The variables of a RK method for a DAE |
| $F$ | Function defining the nonlinear system of an IRK method for an ODE |
| $G$ | Function defining the nonlinear system of an IRK method for a DAE |
| $M$ | Jacobian of the system of an IRK method |
| $N$ | Jacobian of the system of algebraic equations |
| $L$ | Number of Newton iterations |

# Chapter 1

# Introduction

## 1.1 Motivation

Within the team of M. Diehl, an implementation of code generation for Nonlinear Model Predictive Control (NMPC) and Moving Horizon Estimation (MHE) is developed that uses carefully designed numerical methods in order to result in efficient C-code that can be deployed on embedded control systems [25, 58]. The use of efficient, real-time algorithms for the simulation of the model is also very important here. And depending on the used integration method, different types of models will be able to be handled. The work described in this text is focused on this simulation aspect of embedded optimization.

The implementation is carried out in the ACADO code generation tool in which previously only the explicit Runge-Kutta integrator of order 4 (ERK4) with sensitivity generation was available. The use of an explicit integrator, however, can cause problems when using a stiff model for the studied system. The first motivation is therefore that the suite of available integrators for code generation in ACADO should be extended with implicit integrators with efficient sensitivity generation. This would strongly improve the support for stiff systems of equations.

A second motivation is that Runge-Kutta (RK) integrators with continuous output are a promising direction to further accelerate the MHE algorithm and to further widen its application area. Such RK integrators can provide trajectory values between the grid points, which allows them to use larger integration steps than the measurement frequency would imply. This continuous output would make it thus possible to use noisy high frequency data without losing any information, in contrast to averaging these data. A continuous extension to the ERK method is used in [38] for the integration of implicitly discontinuous ODE systems. Implicit RK (IRK) integrators with continuous output already exist, but not using code generation with efficient derivative generation.

A final motivation is that implicit integration methods can often be extended natu-

rally to systems of Differential Algebraic Equations (DAE) instead of only Ordinary Differential Equations (ODE). DAE systems of high index appear in all branches of engineering, in particular mechanical and mechatronic systems often lead to index-3 DAEs. After index reduction, a DAE of index 1 still needs to be simulated by an efficient numerical method [8].

## 1.2 Real-Time Optimization

This section will discuss the solution of real-time estimation and control problems using respectively the principles of Moving Horizon Estimation (MHE) and Nonlinear Model Predictive Control (NMPC), described in [46, 52, 55]. These two optimization strategies both use moving horizon formulations, based on the same optimal control structure [13]. Typically, an estimation and control algorithm (e.g. MHE and NMPC) will interact in an estimation-based feedback control framework as is illustrated in Figure 1.1 [40].

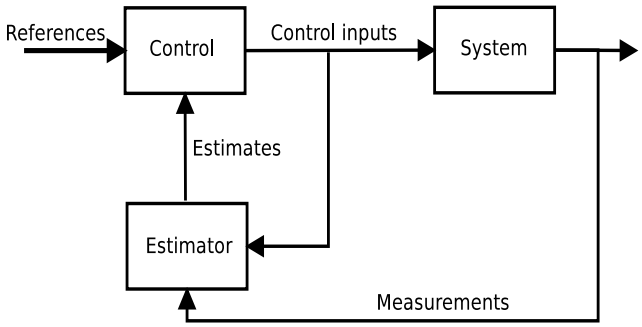



Figure 1.1: Estimation-based Feedback Control: This figure illustrates the interaction between the system and the estimation (e.g. MHE) and control algorithm (e.g. MPC).

In this framework some variables of the system are measured at a certain frequency and based on these measurements and the applied control inputs, the current state of the system (and possibly certain parameters) will be estimated. This can be done using model-based MHE as will be discussed in Section 1.2.2. Given the current state of the system together with all the constraints and objectives, the next 'optimal' value for the control input can be determined and applied to the system using (N)MPC. Real-time estimation and control based on optimization can be very interesting because of the flexibility in formulating the objectives and the model, the capability to directly handle all the constraints, and the possibility to treat disturbances fast. It has also become more and more feasible over the past decade for different applications because of the combination of faster computers and the development of dedicated real-time optimization algorithms for NMPC and MHE.

### 1.2.1 Nonlinear Model Predictive Control

Both MPC and MHE are model-based strategies, i.e. a model for the studied system needs to be available. The following controlled, dynamic ODE system in continuous time will be used on a horizon $[0, T]$:

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

with the continuous time $t$, the controls $u(t)$ and the states $x(t)$. It is assumed that $f$ is either continuous or has finitely many discontinuities with respect to $t$ and that $f$ is Lipschitz continuous with respect to $x$. In that case, a corresponding initial value problem (IVP) has a unique solution [16]. A simplified formulation in continuous time of the Optimal Control Problem (OCP) that needs to be solved for MPC is the following

$$\begin{aligned} \underset{x(t),u(t)}{\text{minimize}} \quad & \int_0^T \|F(t, x(t), u(t))\|_2^2 \, \mathrm{d}t \\ \text{subject to} \quad & x(0) = \bar{x}_0, \\ & \dot{x}(t) = f(t, x(t), u(t)), \\ & 0 = g(x(t), u(t)), \\ & 0 \geq h(x(t), u(t)), \qquad \forall t \in [0, T]. \end{aligned} \tag{1.2}$$

This is an infinite-dimensional optimization problem which will be tackled in Section 1.2.3. A least-squares objective is assumed, because it is preferable for code generation purposes [23]. The MPC approach can now be described as follows:

1. get the current state of the system $\bar{x}_0$ from the estimator (and possibly some parameters)
2. solve approximately the optimization problem in (1.2) and this as fast as possible (to restrain the feedback delay)
3. apply the optimal control solution $u(t)$, $\forall t \in [0, T_s]$ to the system
4. repeat all after the fixed sampling time of $T_s < T$

### 1.2.2 Moving Horizon Estimation

The problem that is solved for MHE is nearly a dual problem of NMPC, because it uses the same optimal control structure. For MHE the controls $u(t)$ are already known because they are provided to the estimator as is also clear from Figure 1.1. The controls $u(t)$ will therefore enter the system dynamics in this case as known, time varying parameters. However, the MHE problem uses a different dynamic model for the system which introduces disturbances $w(t)$ to account for the plant-model mismatch like this:

$$\dot{x}(t) = f(t, x(t), u(t), w(t)), \quad t \in [0, T], \tag{1.3}$$

Based on this model, it is possible to formulate predictions for the different measurements $y(t)$ that are made and forwarded to the estimator (see Figure 1.1). The eventual Optimal Control Problem (OCP) that needs to be solved for MHE has the following form:

$$\begin{aligned} &\underset{x(t),w(t)}{\text{minimize}} \quad \int_0^T \|F(t,x(t),u(t),y(t),w(t))\|_2^2 \,\mathrm{d}t \\ &\text{subject to} \\ &\qquad \dot{x}(t) \;=\; f(t,x(t),u(t),w(t)), \\ &\qquad 0 \;=\; g(x(t),u(t),w(t)), \\ &\qquad 0 \;\geq\; h(x(t),u(t),w(t)), \qquad \forall t \in [0,T]. \end{aligned} \tag{1.4}$$

An estimation for the state of the system is then given by $x(T)$. It is typical for MHE to use a convex function to penalize the mismatch between the real measurements $y(t)$ and the corresponding model predictions in the objective function of the OCP [13]. The disturbances $w(t)$ in the MHE problem take the role of the controls in the NMPC problem and these disturbances are also penalized in the objective function because they represent the plant-model mismatch. The formulation in (1.4) is a simplified one, since usually the objective function also consists of an approximation for the arrival costs [47]. More information on the formulation of the MHE problem can be found in [40]. It is clear that the structure of the MHE problem in (1.4) is very similar to the structure of the NMPC problem in (1.2). The main difference is that for MHE, the control inputs $u(t)$ are given and instead the disturbances $w(t)$ are the new inputs to be optimized, and the initial state $x(0)$ is free. This is also illustrated in Figure 1.2 by showing the estimation horizon used in MHE and the prediction horizon used in NMPC, assuming they have sizes $N_e$ and $N_p$, respectively, and $k$ is the current time. The estimation horizon looks to the past of the system, while the prediction horizon looks to the future of the system.

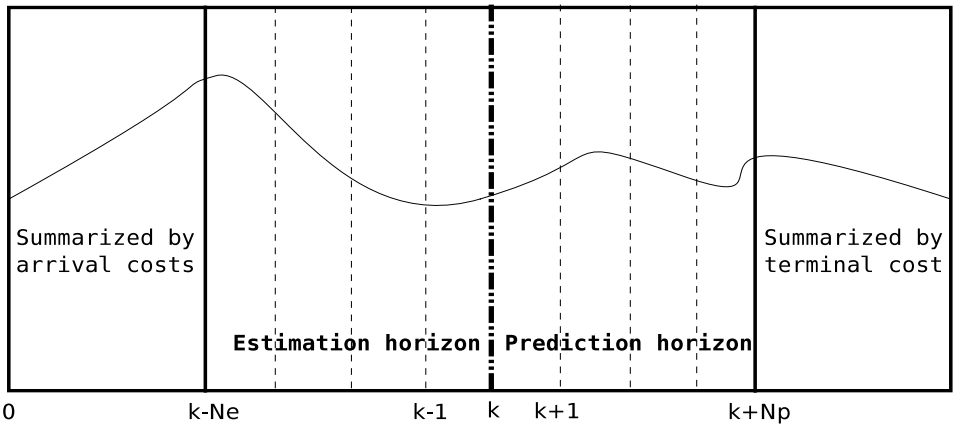



Figure 1.2: Illustration of the estimation horizon (MHE) and the prediction horizon (NMPC), assuming $k$ is the current time.

Another difference is that, after discretization, the optimization variable space for MHE often will be much higher dimensional than for NMPC. This can influence the solvers that should be used for each of these problems. Note that the presented optimization problems are still infinite-dimensional. An approximate solution can then be computed using a numerical method. In what follows, only the OCP formulation of (1.2) is considered which is very similar to (1.4).

### 1.2.3 Optimal Control Algorithms

Most algorithms are based on direct methods which first transform the infinite-dimensional OCP from (1.2) into a finite-dimensional problem. This can be done using a suitable parameterization of the control inputs $u(t)$. There are many ways to do this, but mostly a piecewise constant control parameterization is used. Depending on the discretization of the rest of the problem, a sequential or simultaneous direct method is obtained. A sequential approach is based on the fact that the state trajectory results uniquely from the IVP of (1.1) with $x(0) = x_0$. Let us assume that the horizon $[0, T]$ is discretized using $N$ shooting intervals, resulting in the variables $(x_0, x_1, \ldots, x_{N-1}, x_N)$ and $(u_0, u_1, \ldots, u_{N-1})$. Only the variables $x_0, u_0, u_1, \ldots, u_{N-1}$ are kept as optimization variables. This method, also known as direct single shooting, performs system simulation and optimization sequentially in each iteration. For the simulation part, integrators are needed as presented in this text. The specific structure of the OCP does not play a huge role here, because the reduced problem can often be solved by off-the-shelf optimization code [13].

The simultaneous approach, however, addresses the OCP problem in its full variable space $(x, u)$ directly using a Newton type optimization algorithm. This means that simulation and optimization are performed simultaneously in the form of direct collocation or multiple shooting (MS) methods [16]. In this case it is very important to exploit the specific problem structure. Typically the Newton type optimization procedure shows faster local convergence rates for this simultaneous approach, especially in the case of unstable or highly nonlinear systems. Intuitively, this is because the nonlinearity for the simultaneous approach is equally distributed over the nodes [13]. From the OCP formulation in (1.2), the following discretized MS formulation can be derived:

$$\begin{aligned}
\underset{x,u}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \|F_i(x_i, u_i)\|_2^2 + \|F_N(x_N)\|_2^2 \\
\text{subject to} \quad & x_0 - \bar{x}_0 = 0, \\
& x_{i+1} - f_i(x_i, u_i) = 0, && i = 0, \ldots, N-1, \\
& g_i(x_i, u_i) = 0, && i = 0, \ldots, N-1, \\
& h_i(x_i, u_i) \leq 0, && i = 0, \ldots, N-1,
\end{aligned} \tag{1.5}$$

with $x_i \in \mathbb{R}^{N_x}$ the differential states and $u_i \in \mathbb{R}^{N_u}$ the control inputs. This finite-dimensional problem can then be solved directly using a Newton type optimization algorithm. Since a least-squares objective has been assumed, the Gauss-Newton

method would be preferable here [16]. Note that the equations $x_{i+1} = f_i(x_i, u_i)$ denote the simulation of the ODE model in (1.1), which is the task of the integrator. At this stage, any integrator could still be used including a variable-step variable-order method. For real-time applications, the code for the integration over one shooting interval however needs to be fixed. When the equations of this fixed integrator are included in the problem formulation, a collocation formulation of the OCP is obtained:

$$\begin{aligned}
\underset{x,z,u}{\text{minimize}} \quad & \sum_{i=0}^{N-1} \|\tilde{F}_i(x_i, z_i, u_i)\|_2^2 + \|\tilde{F}_N(x_N)\|_2^2 \\
\text{subject to} \quad & x_0 - \bar{x}_0 = 0, \\
& x_{i+1} - \tilde{f}_i(x_i, z_i, u_i) = 0, \qquad i = 0, \ldots, N-1, \\
& \tilde{g}_i(x_i, z_i, u_i) = 0, \qquad i = 0, \ldots, N-1, \\
& \tilde{h}_i(x_i, z_i, u_i) \leq 0, \qquad i = 0, \ldots, N-1.
\end{aligned} \tag{1.6}$$

This can be regarded as the fully discretized OCP with $z_i \in \mathbb{R}^{N_z}$ the collocation variables or the algebraic states. The free variables in this optimization problem are the differential state vector $(x_0, x_1, \ldots, x_{N-1}, x_N)$, the algebraic state vector $(z_0, z_1, \ldots, z_{N-1})$ and the control vector $(u_0, u_1, \ldots, u_{N-1})$. One shooting interval from time point $k$ to $k+1$ is illustrated in Figure 1.3. A fixed step integrator is used to simulate the system and its variables are contained in $z_k$. The functions $\tilde{g}_i$ define the former equality constraints as well as the collocation equations. It is assumed that the variables $z_i$ are uniquely determined by the values of $x_i$ and $u_i$. Note that this is the same formulation as the one used in [13]. Interesting about this general formulation is that it describes all forms of one-step integration schemes such as explicit and implicit RK methods for ODE as well as DAE systems of equations. Reduction of this fully discretized OCP would lead us back to the formulation in (1.5). It is now useful to give a brief introduction to Newton type optimization.

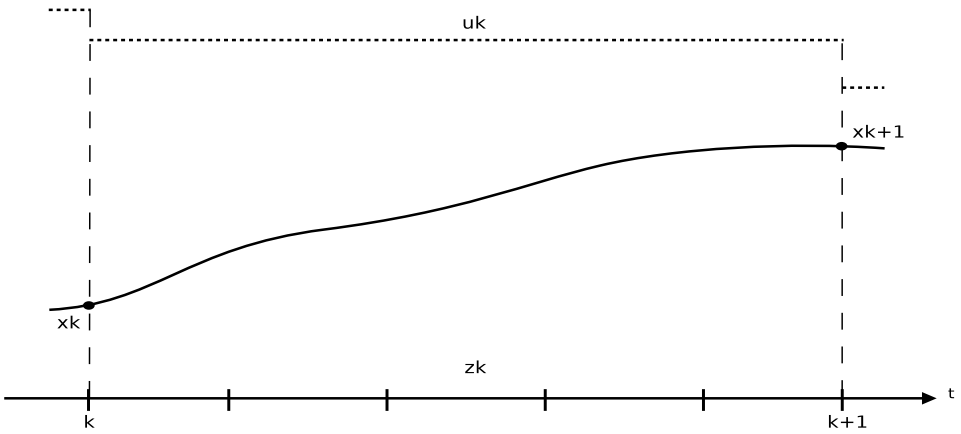



Figure 1.3: Illustration of one shooting interval from time point $k$ to $k+1$, assuming multiple fixed integration steps are taken.

### Newton Type Optimization

Newton type optimization algorithms are the generalization of Newton type methods to nonlinear optimization. Newton's method has locally a quadratic convergence rate, but Newton type methods have slower convergence rates because they do not compute the Jacobian (or its inverse) exactly. The problems in (1.5) and (1.6) are structured cases of a general Nonlinear Program (NLP):

$$\underset{X}{\text{minimize}}\ F(X)\ \text{ s.t. }\ \begin{cases} G(X) = 0 \\ H(X) \leq 0 \end{cases} \tag{1.7}$$

The Karush-Kuhn-Tucker (KKT) conditions need to be satisfied for a locally optimal solution $X^*$ of this problem [15]. A Newton type optimization method then tries to find a solution of the system of KKT conditions using successive linearizations of the problem function. The two big families of such methods are the Sequential Quadratic Programming (SQP) and the Interior Point (IP) methods. Their major difference lies in the way they treat the KKT condition, corresponding to the inequality constraints $H(X) \leq 0$ in (1.7). More information on both families of Newton type optimization methods can be found in the references, already mentioned in this section.

Because especially SQP type methods are implemented in the ACADO Toolkit, their general idea will be briefly described here. This approach to handle an NLP with inequalities is based on the quadratic model interpretation of Newton type methods [16]. An SQP method iteratively solves the system of KKT conditions by linearizing all nonlinear functions that appear. The resulting linear system can be interpreted as the KKT conditions of the following Quadratic Program (QP):

$$\begin{aligned} &\underset{X}{\text{minimize}}\ F^k_{\text{QP}}(X)\ \text{ s.t. }\ \begin{cases} G(X^k) + \nabla G(X^k)^{\text{T}}(X - X^k) = 0 \\ H(X^k) + \nabla H(X^k)^{\text{T}}(X - X^k) \leq 0 \end{cases} \\ &F^k_{\text{QP}}(X) = \nabla F(X^k)^{\text{T}} X + \frac{1}{2}(X - X^k)^{\text{T}} \nabla^2_X \mathcal{L}(X^k, \lambda^k, \mu^k)(X - X^k) \end{aligned} \tag{1.8}$$

When such a QP problem is solved in every iteration, the method is called an 'exact Hessian' SQP method. More widely used SQP variants, however, do not use the exact Hessian matrix $\nabla^2_X \mathcal{L}(X^k, \lambda^k, \mu^k)$ but use approximations, such as in Powell's Classical SQP method and the Constrained Gauss-Newton method [13, 15, 16]. From this QP formulation it is also clear why the used integrator should generate the sensitivities with respect to the variables (discussed in Chapter 3) to be able to evaluate the different Jacobian matrices.

## 1.3 The ACADO Toolkit

ACADO Toolkit is a software environment and algorithm collection written in C++ for automatic control and dynamic optimization. It is an open source project developed at K.U. Leuven. It provides a general framework for using a great variety of

algorithms for direct optimal control, including MPC as well as state and parameter estimation. It also provides (stand-alone) efficiently implemented RK and Backward Differentiation Formulas (BDF) integrators for the simulation of ODE and DAE systems. The very intuitive syntax of ACADO allows the user to formulate control problems in a way that is very close to the usual mathematical syntax. There is also a Matlab interface with the ACADO Toolkit available.

The ACADO Toolkit has been designed to meet the following four key properties: open-source package, user-friendly, extensible code and self-contained [35]. The current version of the ACADO Toolkit supports the following 4 problem classes:

- OCP problems: off-line dynamic optimization problems
- Multi-objective optimization and optimal control problems
- Parameter and state estimation problems
- MPC problems and online state estimation

More information on the use and the possibilities of the ACADO Toolkit and ACADO for Matlab can be found in the ACADO Toolkit User's Manual [36] and the ACADO for Matlab User's Manual [22]. More information on the design of the ACADO Toolkit can be found in [35].

## 1.4 Online Algorithms

It would be the dream in NMPC and MHE to have the solution to the next OCP instantly, which is of course impossible because of computational delays. There are, however, online algorithms available to deal with this issue such as described in [3, 37, 39]. A clear survey and classification of some of these algorithms can be found in [13]. Such algorithms are mostly based on the following ideas:

- It can be beneficial to do as many computations offline as possible and this can go from code optimizations to a complete precomputation of the control law in some cases.
- Instead of starting e.g. an NMPC problem at the current state of the system, it can be a good idea to simulate at which state the system will be when the problem will be solved. This way, the delay can be compensated.
- The needed computations in each sampling time can be divided into a preparation and a feedback phase. The preparation phase is then the most CPU intensive one and the feedback phase will quickly deliver an approximate solution to the OCP.
- Instead of performing iterations for an NMPC problem that is getting older and older, it is also possible to work with the most recent information in each new iteration.

### 1.4.1 The Real-Time Iteration Scheme

The Real-Time Iteration (RTI) Scheme is one of the online algorithms described in [13] and is presented in more detail in [17]. Note that these references focus on the NMPC problem but the idea is also transferable to the MHE problem, which is discussed for example in [18]. This is the only online algorithm that will briefly be introduced here, because it is used in the ACADO Toolkit [35]. It performs only one SQP type iteration with Gauss-Newton Hessian per sampling time.

The scheme uses the idea to divide the computations into a preparation and a feedback phase. The preparation phase takes the longest and will take care of the linearization of the system, the elimination of algebraic variables and condensing of the linearized subproblem. This means that the feedback phase only needs to solve one condensed QP problem and is therefore much shorter. This feedback phase can even be orders of magnitude shorter than the preparation phase [35]. Note that the preparation phase can only make use of the expected state $\bar{x}_0$ for the NMPC problem while this will be replaced by the actual state $\bar{x}_0'$ in the feedback phase. The RTI scheme is more than just doing only one SQP iteration per sampling time, the principle of initial value embedding namely provides extra performance which is basically for free [13, 14].

## 1.5 Code Generation

This section will discuss the idea to automatically generate tailored source code in order to speed-up the numerical solution of optimization problems. Figure 1.4 illustrates this principle of code generation. The idea is to obtain a custom solver based on the description of a certain problem. This custom solver can then be used to efficiently solve specific instances of this problem.

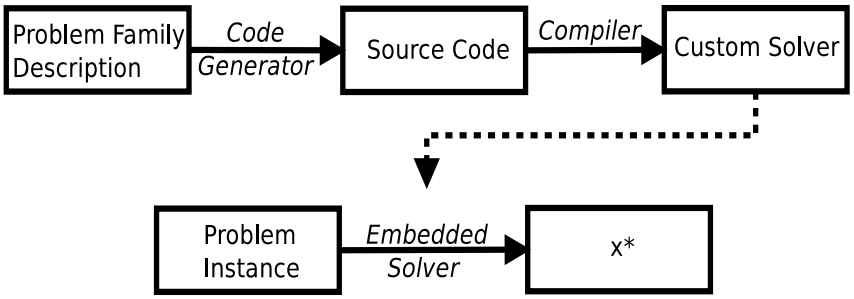



Figure 1.4: This figure illustrates the principle of code generation to obtain a custom solver for a certain problem family [45].

More than 20 years ago, a code generation environment was already presented in [50] to export tailored implementations of Karmarkar's algorithm for solving Linear Programs (LP). About one decade ago, the software *AutoGenU* was developed by Ohtsuka and Kodama which provided code generation for nonlinear MPC [48, 49].

More recently, code generation has attracted great attention due to the software package *CVXGEN* [44], which can be tested by academic users via the web interface [43]. The Multi-Parametric Toolbox (MPT) for example also provides code generation for explicit MPC [42]. More information on approaches to automatic code generation for the numerical solution of optimization problems can be found in [23].

Let us briefly discuss the advantages that motivate code generation. The first reason to use code generation is because it can speed-up the computations by removing unnecessary computations as well as by optimization of the generated source code. The latter consists of an optimized memory management because problem dimensions and sparsity patterns can be detected and hard-coded, a more efficient cache usage by reorganizing the computations and other techniques like loop unrolling. The second reason is the increased adaptivity of the numerical algorithm that will be exported. When implementing code generation, it is namely very easy to offer options to e.g. choose between single and double precision arithmetic, avoid the use of certain standard libraries that are unavailable on the target hardware or e.g. to choose the programming language in which the optimization code will be exported [23].

These motivations to use code generation are clearly more relevant to real-time optimization of fast systems such as they appear in robotics and mechatronics, because of the very high sampling rates. Real-time optimization algorithms are also often run on embedded hardware imposing extra restrictions which can be taken into account when exporting the source code.

### 1.5.1 ACADO Code Generation

The ACADO code generation tool is written as an add-on to the ACADO Toolkit (Section 1.3) and it exports efficient C-code solving nonlinear MPC/MHE problems by means of the RTI scheme with Gauss-Newton Hessian approximation. The exported code is self-contained and efficient because it is tailored to the specific problem formulation using the symbolic ACADO syntax. The discretization of the time-continuous system is possible using single or multiple shooting with the use of an equidistant or non-equidistant grid. The resulting large but sparse QP problem is first reduced to a smaller-scale but dense QP. This smaller-scale QP is then solved at each sampling instant using an embedded variant of *qpOASES* or a dense QP solver exported by *CVXGEN*.

The real-time iteration algorithm is based on a shooting discretization of the OCP, which means that the integration of the ODE/DAE system and the computation of the first-order derivatives with respect to the variables are a main computational step [23]. In order to guarantee a deterministic runtime of the online integrator, only fixed step methods can be used for code generation. This means that a carefully made choice for this step size is needed. At the start of this project the only available integrator for code generation was the ERK4 method using the variational differential

equations to generate the needed derivatives (see Chapter 3). The methods that will be presented in this text, are however added to the suite of integrators available to the code generation tool of ACADO.

The ACADO code generation tool can speed-up the computations in various ways. An important first observation is that all dimensions can be hard-coded, completely avoiding dynamic memory allocation. The auto-generated source code therefore only uses static global memory which also ensures that no segmentation faults or out-of-memory errors can occur. Moreover, using loop unrolling it is possible to achieve a more linear code without too many conditional statements. Besides the increased efficiency, avoiding conditional statements can also reduce the chance of running into a part of the code which has accidentally never been tested. Some operations can also be made more efficient by hard-coding known numerical values or by exploiting sparsity patterns.

More information on the capabilities of the ACADO code generation tool can be found in [25, 36, 58]. The tool has been applied to a simplified model of a kite carousel in [24] of which the results are summarized in [23]. The following table shows the worst-case timing results of all algorithmic components for a complete real-time iteration.

| | CPU time | % |
|---|---|---|
| Integration and sensitivity generation | 0.59 ms | 78 % |
| Condensing | 0.09 ms | 12 % |
| QP solution (using qpOASES) | 0.04 ms | 5 % |
| Remaining operations | 0.04 ms | 5 % |
| One complete real-time iteration | 0.76 ms | 100 % |

Table 1.1: Worst-case timing results of the auto-generated real-time iteration algorithm applied to the kite carousel model [23].

From this table it is clear that the major share of the computation for this example was spent within the auto-generated code for the ERK4 integrator. The fact that the integration of the system with the generation of the sensitivities forms a big computational step in the real-time iteration scheme is also a very important motivation for this project. The reason for this is described by Amdahl's law which states that the speed-up achievable from a certain improvement is limited by the proportion of the computations that is affected by this improvement. Assume for example that for the integration method, which is a proportion $P = 0.78$ of the computations (see Table 1.1), a speed-up would be possible of $S = 2$ (twice as fast). Amdahl's law then states that the overall speed-up of a complete real-time iteration by applying this improvement in only the integration method would be:

$$\frac{1}{(1-P)+\frac{P}{S}} = \frac{1}{0.22+0.39} = \frac{1}{0.61} \approx 1.64 \tag{1.9}$$

## 1.6 Contents

The text will start by introducing the Runge-Kutta and collocation methods and their properties in chapter 2. The chapter will also cover some important implementation aspects. Chapter 3 then discusses in more detail different ways for the efficient generation of the sensitivities by the integration methods. The extension of all these methods to DAE systems is the subject of Chapter 4 and Chapter 5 will elaborate on the implementation of the continuous output feature. It is however an important contribution of this thesis that the presented methods are also implemented and made available as open source code. This allows us to present numerical experiments in Chapter 6 which confirm claims made during the text and support the final implementation in the ACADO Toolkit.

# Chapter 2

# Integration methods

In this chapter, the emphasis will lie on the integration of a system of ODEs. The treatment of DAEs will be discussed in Chapter 4. So, the assumption in this chapter is that the following Initial Value Problem (IVP) needs to be solved over some time interval:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)), \\ x(0) &= x_0, \end{aligned} \tag{2.1}$$

where $x(t)$ is a vector with the $N_x$ differential states at time $t$ and $u(t)$ is a vector with the $N_u$ control inputs at time $t$, which are assumed to be known here. Because the control inputs $u$ are known at every time $t$, the IVP can also be written as follows:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t)) \\ x(0) &= x_0 \end{aligned} \tag{2.2}$$

The chapter will start by introducing the Runge-Kutta methods in Section 2.1 and by explaining their connection with the collocation methods in Section 2.2. Section 2.3 then discusses stiff systems of equations and the desirable stability properties for integration methods to be able to handle them efficiently. Eventually, the major implementation aspects are addressed in Section 2.4 and 2.5. They respectively discuss solving the nonlinear system of an IRK method and the initialization of the variables.

## 2.1 Runge-Kutta Methods

It can be useful to give a very short introduction to the general class of Linear Multistep (LM) methods first. A linear $k$-step method uses a linear combination of the $k$ previous solution points and derivative values. The following formulation of a LM method with $k$ steps will be used [19]:

$$x_{n+k} = h \sum_{j=0}^{k} \beta_j f_{n+j} - \sum_{j=0}^{k-1} \alpha_j x_{n+j}, \tag{2.3}$$

where $\alpha_j$ and $\beta_j$ are the constants defining the method, with at least $\alpha_0$ or $\beta_0$ different from zero. Note that in this equation the variable $x_{n+j}$ is the approximation of the solution $x(t)$ at time $t_{n+j} = t_n + jh$ and the variable $f_{n+j}$ denotes the corresponding evaluation of the function $f(t_{n+j}, x_{n+j})$. Such a LM method is called explicit when $\beta_k = 0$ and implicit when $\beta_k \neq 0$. In the latter case the variable $x_{n+k}$, which needs to be calculated, appears in the left- as well as in the right-hand side of (2.3). This means that an iterative process will be needed. More information on these LM methods and on their properties can be found in [19].

Unlike LM methods, single-step methods refer to only one previous solution point and its derivative to determine the current value. The family of RK methods is the most important class of single-step methods that are generically applicable to ODEs. LM methods achieve their high order by doing more than one step, while they maintain the linearity with respect to the values $x_{n+j}$ and $f_{n+j}$, $j = 0, 1, \ldots, k$. The idea behind the RK methods however is to increase the order by abandoning this linearity, while they remain single-step methods. Instead of using more than one previous solution point, a RK method uses only one previous point and some stage values that can be viewed as intermediate values of the solution $x(t)$ at the times $t_n + c_i h$. These values are computed within each integration step. The number of stages $s$ of a RK method is the number of stage values that are used and the values are denoted by $X_i$. Eventually, the formulation of a RK method with $s$ stages is the following [28]:

$$\begin{aligned} X_1 &= x_n + h \sum_{j=1}^{s} a_{1j} f(t_n + c_j h, X_j), \\ &\ldots \\ X_s &= x_n + h \sum_{j=1}^{s} a_{sj} f(t_n + c_j h, X_j), \\ x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i f(t_n + c_i h, X_i), \end{aligned} \tag{2.4}$$

with $b_i$ $(i = 1, \ldots, s)$ the weights and $a_{ij}$ the internal coefficients, which satisfy the internal consistency condition:

$$c_i = \sum_{j=1}^{s} a_{ij} \tag{2.5}$$

For the method to have at least order 1, the variables $b_i$ must satisfy

$$\sum_{i=1}^{s} b_i = 1 \tag{2.6}$$

This is the first one of the order conditions. Any RK method that satisfies this condition is consistent and thus also convergent (assuming some other conditions hold) [28]. The formulas in (2.4) are often represented using a Butcher table:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{2.7}$$

### 2.1.1 Explicit RK Methods

Explicit RK methods have the property that the expressions for their stage values $X_i$ are explicit, which means that they only depend on the previous solution point $x_n$ and the previously computed stage values $X_j$ with $j = 1, \ldots, i-1$. By looking at the formulas in (2.4) and the Butcher table in (2.7), it should be clear that for an explicit RK method the matrix $A = (a_{ij})$ is strictly lower triangular. The simplest RK method is the forward Euler method, it is also the only consistent explicit RK method with one stage ($s = 1$). The corresponding Butcher table looks like this:

$$\begin{aligned} x_{n+1} &= x_n + hf(t_n, x_n) \\ \begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \end{aligned} \tag{2.8}$$

Another important, explicit RK method of order 4 has 4 stages and the following Butcher table:

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} \tag{2.9}$$

This RK method is so commonly used that it is often referred to as RK4 or *the* ($4^{th}$ order) RK method. Note that this was the only integration method available in the code generation tool of the ACADO Toolkit at the start of this project. In what follows, this RK method will be denoted by ERK4 to emphasize the fact that this is an explicit method.

### 2.1.2 Implicit RK Methods

Implicit RK methods are methods for which the matrix $A = (a_{ij})$ is not strictly lower triangular. To be able to calculate the stage values $X_i$, a system of $N_x \times s$ nonlinear equations and unknowns needs to be solved. This can be done using the Newton method or a variant of this method in which the Jacobian (or the inverse of the Jacobian) is approximated. This clearly is a major disadvantage, compared to the explicit RK methods. Among the advantages of using an implicit RK method is for example that there always exists a method with $s$ stages that has an order $2s$, while there is no explicit RK method with $s$ stages that has an order larger than $s$ [19]. Another very important advantage of implicit RK methods are their better

stability properties, which are especially important in the case of stiff problems. This will be clarified in Section 2.3.

The simplest example of an implicit RK method is the backward Euler method:

$$x_{n+1} = x_n + hf(t_n + h, x_{n+1})$$

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{2.10}$$

Another important, implicit RK method is the trapezoidal rule which is also known as the Crank-Nicolson method:

$$x_{n+1} = x_n + \frac{h}{2}(f(t_n, x_n) + f(t_{n+1}, x_{n+1}))$$

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array} \tag{2.11}$$

More examples of implicit RK methods will follow in the next section.

### 2.1.3 Semi-implicit RK Methods

In addition to the explicit and implicit methods, there are also semi-implicit RK methods. These methods are for example studied in [4] and [7]. They are not explicit, which means that the matrix $A = (a_{ij})$ is not strictly lower triangular but they are also not fully implicit. In the case of a semi-implicit RK method, the matrix $A$ exhibits a specific structure which allows one to lower the computational cost. Implicit RK methods have one large nonlinear system that needs to be solved iteratively. For Diagonal Implicit RK (DIRK) methods every stage, however, leads to a nonlinear system that can be solved separately in the right order. The coefficient matrix $A$ is then lower triangular, as is depicted in Figure 2.1. While this strongly lowers the computational cost, these methods can still have good stability properties. When DIRK methods have identical diagonal elements in the matrix $A$ (all equal to $\gamma$ in Figure 2.1), they are called Singly DIRK (SDIRK) methods. For the nonlinear system corresponding each stage, the same approximation of the Jacobian can then be reused for the Newton iterations. This avoids extra matrix evaluations and LU factorizations which can be costly. Eventually, an explicit SDIRK (ESDIRK) method has an explicit first stage as is also illustrated in the figure below.

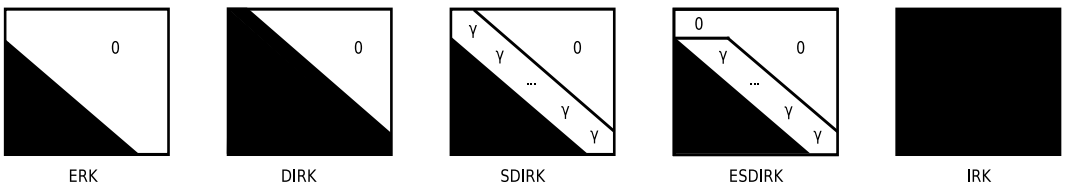



Figure 2.1: Structure of the matrix $A$ for different families of RK methods [41].

An interesting example of a semi-implicit RK method is the four-stage ESDIRK method from [41] with a Butcher table that is structured as follows:

$$\begin{array}{c|cccc} 0 & 0 & & & \\ c_2 & a_{21} & \gamma & & \\ c_3 & a_{31} & a_{32} & \gamma & \\ 1 & b_1 & b_2 & b_3 & \gamma \\ \hline & b_1 & b_2 & b_3 & \gamma \end{array} \tag{2.12}$$

This method has order 3, is A- and L-stable (see Section 2.3.2) and is stiffly accurate. The latter means that the last stage value is equal to the solution at the end of the current integration step. This can also be seen in (2.12), since $c_4 = 1$ and $a_{4i} = b_i$ for $i = 1, \ldots, 4$. Since the method also has an explicit first stage, only 3 of the 4 stages need to be computed during each step. The first stage is namely equal to the last stage of the previous step. Therefore, this is also known as the first-same-as-last (FSAL) design. The exact values for the coefficients in the Butcher table of this method can be found in [41].

## 2.2 Collocation Methods

The following discussion on collocation methods is based on the one in [27]. A natural way to construct a single-step method is to start by integrating both sides of the ODE over one step $h$

$$x(t+h) = x(t) + \int_t^{t+h} f(\tau, x(\tau))\,\mathrm{d}\tau \tag{2.13}$$

The integral that remains can be approximated using a numerical quadrature rule. For example, it can be approximated by exactly integrating an interpolating polynomial. This is what is done for the collocation methods. Using $s$ quadrature points $c_i$ which satisfy $0 \leq c_1 < c_2 < \ldots < c_s \leq 1$ (distinct nodes on the unit interval), a collocation polynomial $p(t)$ of degree $s$ can be defined. This polynomial needs to satisfy the following $s + 1$ conditions:

$$\begin{aligned} p(t_n) &= x_n \\ \dot{p}(t_n + c_i h) &= f(t_n + c_i h, p(t_n + c_i h)) \quad \text{for} \quad i = 1, \ldots, s. \end{aligned} \tag{2.14}$$

These conditions express that the polynomial starts at $(t_n, x_n)$ and satisfies the ODE at the $s$ nodes on $[t_n, t_{n+1}]$. The numerical result of a collocation method for the solution of the ODE at time $t_{n+1} = t_n + h$ is then simply the value of this polynomial at that time: $x_{n+1} = p(t_n + h)$. This method can now be written in the form of an implicit RK method. All collocation methods are implicit RK methods, but not all implicit RK methods are collocation methods. To show this, the formulas in (2.4) will first be rewritten using the variables $k_i = f(t_n + c_i h, X_i)$ instead of the stage values $X_i$. This results in the following formulation:

$$\begin{aligned} k_1 &= f(t_n + c_1 h, x_n + h \sum_{j=1}^{s} a_{1j} k_j) \\ &\dots \\ k_s &= f(t_n + c_s h, x_n + h \sum_{j=1}^{s} a_{sj} k_j) \\ x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i k_i \end{aligned} \tag{2.15}$$

For a collocation method, the polynomial that satisfies the conditions in (2.14) is used to define these variables $k_i$ like this:

$$k_i = \dot{p}(t_n + c_i h) = f(t_n + c_i h, p(t_n + c_i h)),\ i = 1, \dots, s. \tag{2.16}$$

The basis that consists of the Lagrange interpolating polynomials will be used here to define the polynomial $\dot{p}(t)$:

$$\dot{p}(t) = \sum_{i=1}^{s} k_i l_i \left( \frac{t - t_n}{h} \right) \tag{2.17}$$

Note that the Lagrange interpolating polynomials $l_i$, $i = 1, \dots, s$ for the already mentioned quadrature points $c_i$ are defined as

$$l_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{s} \frac{t - c_j}{c_i - c_j} \tag{2.18}$$

The $i^{th}$ Lagrange interpolating polynomial $l_i$ is equal to 1 for $t = c_i$, but equal to 0 for $t = c_j$ with $j \neq i$. This explains the definition of the polynomial $\dot{p}(t)$. The value of the polynomial $p(t)$ at time $t_n + c_i h$ can now be found by integrating (2.17):

$$p(t_n + c_i h) = p(t_n) + \sum_{j=1}^{s} k_j \int_{t_n}^{t_n + c_i h} l_j \left( \frac{t - t_n}{h} \right) \mathrm{d}t,\ i = 1, \dots, s \tag{2.19}$$

Note that from the conditions in (2.14), it is known that $p(t_n) = x_n$. The equation can also be simplified using the substitution $\tau = \frac{t - t_n}{h}$ and $dt = h d\tau$:

$$p(t_n + c_i h) = x_n + h \sum_{j=1}^{s} k_j \int_{0}^{c_i} l_j(\tau)\, \mathrm{d}\tau,\ i = 1, \dots, s \tag{2.20}$$

Similarly, the value of the collocation polynomial at time $t_n + h$ is described by the following expression:

$$x_{n+1} = p(t_n + h) = x_n + h \sum_{j=1}^{s} k_j \int_{0}^{1} l_j(\tau)\, \mathrm{d}\tau \tag{2.21}$$

The canonical choice for the weights $b_i$ and the internal coefficients $a_{ij}$ is as follows for $i, j = 1, \ldots, s$:

$$\begin{aligned} a_{ij} &= \int_0^{c_i} l_j(\tau)\,\mathrm{d}\tau \\ b_i &= \int_0^{1} l_i(\tau)\,\mathrm{d}\tau \end{aligned} \tag{2.22}$$

Then the formulas of a RK method from (2.15) can be found by applying these new definitions of $b_i$ and $a_{ij}$ to equations (2.21) and (2.20). The resulting Equation (2.20) needs to be substituted into (2.16) and this should clarify that all collocation methods are implicit RK methods. It should also be clear why not all RK methods are collocation methods, because collocation methods impose more restrictions on the coefficients $c_i$, $b_i$ and $a_{ij}$. The coefficients $c_i$, which determine the quadrature nodes, need to be distinct for a collocation method and the coefficients $b_i$ and $a_{ij}$ are then completely defined by these quadrature points (see Equation (2.22)). So the class of RK methods is actually a generalization of the collocation methods which allows these coefficients to take arbitrary values.

### 2.2.1 The Gauss-Legendre collocation methods

From the previous discussion, it is clear that only the quadrature nodes $c_i$ are needed to define a collocation method. The idea in this text will be to choose one type of collocation methods, so the focus can lie more on the implementation in the code generation tool of ACADO and on the different topics discussed in the next chapters. The Gauss-Legendre collocation methods will be used, because they are optimal in terms of order of accuracy [27]. The $s$ nodes are placed at the roots of a Legendre polynomial. The methods corresponding to $s = 1$, 2 and 3 (order $p = 2$, 4 and 6) are used and they have as quadrature points:

$$c_1 = \frac{1}{2}, \qquad s = 1, p = 2, \tag{2.23}$$

$$c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6},\ c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}, \qquad s = 2, p = 4, \tag{2.24}$$

$$c_1 = \frac{1}{2} - \frac{\sqrt{15}}{10},\ c_2 = \frac{1}{2},\ c_3 = \frac{1}{2} + \frac{\sqrt{15}}{10}, \quad s = 3, p = 6. \tag{2.25}$$

For completeness, the Butcher tables for these 3 collocation methods are stated. These can be found easily by using the definition of the Lagrange interpolating polynomials from (2.18) in the expressions for the coefficients in (2.22).

**The 2nd order method:**

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array} \tag{2.26}$$

**The $4^{\text{th}}$ order method:**

$$\begin{array}{c|cc} 1/2+\sqrt{3}/6 & 1/4 & \sqrt{3}/6+1/4 \\ 1/2-\sqrt{3}/6 & -\sqrt{3}/6+1/4 & 1/4 \\ \hline & 1/2 & 1/2 \end{array} \tag{2.27}$$

**The $6^{\text{th}}$ order method:**

$$\begin{array}{c|ccc} 1/2-\sqrt{15}/10 & 5/36 & 2/9-\sqrt{15}/15 & 5/36-\sqrt{15}/30 \\ 1/2 & 5/36+\sqrt{15}/24 & 2/9 & 5/36-\sqrt{15}/24 \\ 1/2+\sqrt{15}/10 & 5/36+\sqrt{15}/30 & 2/9+\sqrt{15}/15 & 5/36 \\ \hline & 5/18 & 4/9 & 5/18 \end{array} \tag{2.28}$$

### 2.2.2 Continuous Output

The collocation methods not only give an approximation for the next solution point, given the previous one. They can provide a continuous approximation of the solution on the time interval $[t_n, t_{n+1}]$. This makes the principle of continuous output possible for integrators, which will be discussed in more detail in Chapter 5. As already mentioned, a collocation method constructs a polynomial $p(t)$ that passes through $(t_n, x_n)$ and that agrees with the ODE system at $s$ nodes on $[t_n, t_{n+1}]$. The value of the polynomial at time $t_n + c_i h$ can be found using (2.20). The value at time $t_n + ch$, with $c \in [0, 1]$, can be found similarly and this results in

$$x(t_n + ch) \approx x_{n+c} = x_n + h \sum_{j=1}^{s} k_j \int_0^c l_j(\tau)\, \mathrm{d}\tau \tag{2.29}$$

In this equation the integrals $\int_0^c l_j(\tau)\, \mathrm{d}\tau$ are again independent of the previous solution point $(t_n, x_n)$ and of the step size $h$. They only depend on the value $c \in [0, 1]$ and on the Lagrange interpolating polynomials from (2.18) which means they depend on the quadrature points $c_i$. The approximation $x_{n+c}$ of the solution at time $t_n + ch$ for the 2 first Gauss-Legendre methods from the previous section can for example be obtained using the equations:

$$x_{n+c} = x_n + hk_1 c, \qquad (s = 1, p = 2) \tag{2.30}$$

$$x_{n+c} = x_n + h(k_1 \frac{c}{2}(\sqrt{3}c + 1 - \sqrt{3}) + k_2 \frac{c}{2}(-\sqrt{3}c + 1 + \sqrt{3})), \ (s = 2, p = 4) \tag{2.31}$$

Note that the variables $k_i$ for $i = 1, \ldots, s$ are here assumed to be known, after solving the system of nonlinear equations from (2.15).

### 2.2.3 The order of accuracy

Kuntzmann (1961) and Butcher (1964) discovered that for every value of $s$ (the number of quadrature points), there exists an IRK method of order $2s$ which is optimal in terms of order of accuracy. The Gauss-Legendre collocation methods are

already presented as such IRK methods of order $2s$ and the proof can be found in [32].

It may however happen that the order $p^*$ of the continuous output on the interval $[t_n, t_{n+1}]$ is smaller than the order $p$ of the method at the quadrature points. The continuous output is obtained as an interpolating polynomial $p(t)$ of degree $s$ and defined by the $s+1$ conditions in (2.14). This means the interpolation error for this collocation polynomial is $O(h^{s+1})$ or the order of the continuous output is $p^* = s+1$. The following table summarizes the order of accuracy for the 3 Gauss-Legendre methods:

| number points ($s$) | order method ($p = 2s$) | order output ($p^* = s+1$) |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 4 | 3 |
| 3 | 6 | 4 |

The effect of the lower order in between the quadrature points clearly is more important for higher order Gauss methods. Figure 2.2 shows a typical plot of the mean relative error with respect to the step size $h$ for the $4^{\text{th}}$ order Gauss-Legendre method. The figure illustrates that in this case the order in between the quadrature points is one lower ($p^* = 3$) than the order at these points ($p = 4$).

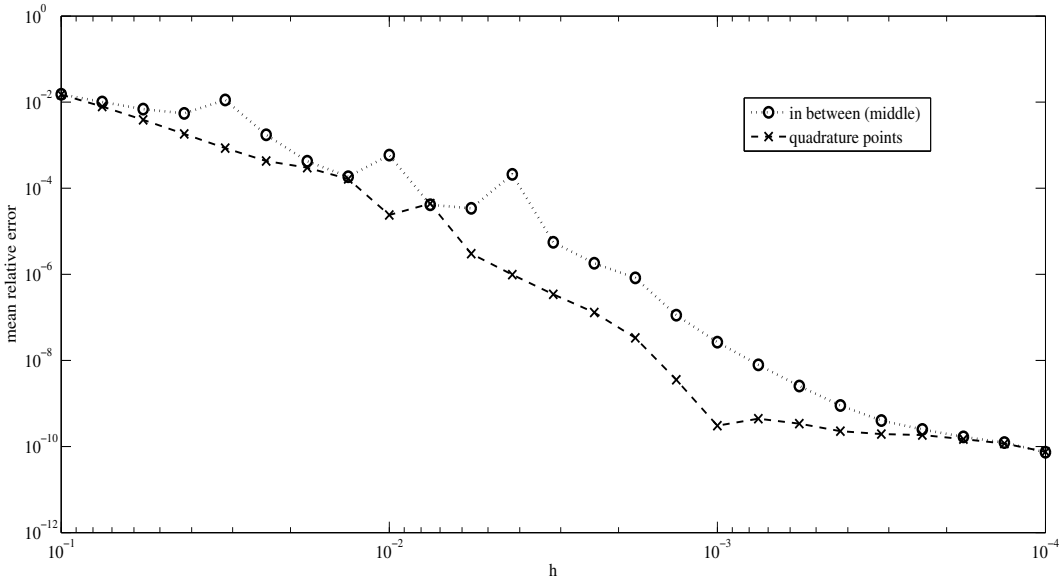



Figure 2.2: An example of a comparison between the relative error at the quadrature points and in the middle of these points with respect to the step size $h$ for the $4^{\text{th}}$ order Gauss method.

## 2.3 Stiff systems of equations

As can clearly be seen from the Butcher tables in the previous section, the used integrators are implicit RK methods instead of explicit ones. One of the reasons to use implicit RK methods is their better stability. They will perform much better on stiff problems, which are quite common in real-world applications. It is not one of the goals of this section to give an extensive set of examples of stiff problems or of methods which perform well on stiff systems. This is covered by many good books such as [33]. A short introduction on what defines a system to be stiff will however be given for the linear and nonlinear case. This will lead to the formulation of the most important desired stability properties and then the 3 Gauss-Legendre collocation methods from the previous section can be discussed in terms of these properties.

### 2.3.1 Characterization of stiffness

A good definition of stiffness is given by J. D. Lambert. He said a system is stiff in a certain interval if a numerical method with a finite region of absolute stability, applied to that system, is forced to use a step length which is excessively small in relation to the smoothness of the exact solution in that interval.

**Linear stiff problems:** Let us first have a quick look at linear systems of equations such as

$$\dot{x}(t) = Ax(t) + \phi(t), \tag{2.32}$$

where $A$ is a $m \times m$ matrix with $m$ eigenvalues $\lambda_i$ and corresponding eigenvectors $c_i$. The general solution of this system of equations looks like this:

$$x(t) = \sum_{i=1}^{m} k_i \exp(\lambda_i t) c_i + \psi(t) \tag{2.33}$$

Let us assume that the linear ODE system is stable, which means that all the eigenvalues $\lambda_i$ have a negative real part. The term $\sum_{i=1}^{m} k_i \exp(\lambda_i t) c_i$ in the solution is a transient term which depends on the initial value $x(0)$ and goes to zero for $t \to \infty$. The other term $\psi(t)$ is independent of this initial value and represents the inhomogeneous solution. If $\mathrm{Re}(\lambda_i)$ denotes the real part of the eigenvalue $\lambda_i$, assume the eigenvalues are ordered like this:

$$|\mathrm{Re}(\lambda_1)| \geq |\mathrm{Re}(\lambda_2)| \geq \ldots \geq |\mathrm{Re}(\lambda_m)| \tag{2.34}$$

If the steady state solution $\psi(t)$ needs to be found numerically, then the following can be observed:

- A smaller value of $|\mathrm{Re}(\lambda_m)|$ means that it will take longer to be able to neglect the transient term with respect to the steady state solution. The integration interval then needs to be longer, so it would be desirable to take larger steps.

- A larger value of $|\mathrm{Re}(\lambda_1)|$ means that a smaller step length $h$ will be needed to keep the calculations numerically stable.

This should motivate the following definition of stiffness: a linear ODE system such as in (2.32) is stiff if

$$\begin{cases} \mathrm{Re}(\lambda_i) < 0, \ i = 1, 2, \ldots, m \\ |\mathrm{Re}(\lambda_1)| \gg |\mathrm{Re}(\lambda_m)| \end{cases} \tag{2.35}$$

This is why the stiffness ratio can be defined as

$$\frac{|\mathrm{Re}(\lambda_1)|}{|\mathrm{Re}(\lambda_m)|} \tag{2.36}$$

**Nonlinear stiff problems:** The previous discussion on the stiffness of linear ODE systems can be extended to the nonlinear case. Nonlinear systems like in (2.2) are called stiff when the eigenvalues of the Jacobian $\partial f/\partial x$ show similar characteristics as the eigenvalues of the matrix $A$ of a stiff linear system. Note, however, that these eigenvalues are not constant for a nonlinear system because the Jacobian is not constant either. The eigenvalues depend on the time $t$, so a nonlinear system can be stiff on a certain time interval if the eigenvalues satisfy the conditions from (2.35) on this interval.

### 2.3.2 Desirable stability properties

The discussion on some desired stability properties will start with the definition of the stability function and the region of absolute and relative stability.

**The stability function:** To determine the stability function for a RK method, this method needs to be applied to Dahlquist's equation $\dot{x} = \lambda x$ [12]. The resulting equations need to be solved for the next solution point $x_{n+1}$ in terms of the previous point $x_n$ to give an explicit function like this:

$$x_{n+1} = R(\lambda h)x_n = R(z)x_n \tag{2.37}$$

The stability function is this function $R(z) = R(\lambda h)$ which defines the relation between the previous and the next solution point. The stability function of a collocation method, based on the points $c_1, c_2, \ldots, c_s$, is given by [33]

$$R(z) = \frac{M^s(1) + M^{s-1}(1)z + \ldots + M(1)z^s}{M^s(0) + M^{s-1}(0)z + \ldots + M(0)z^s} = \frac{P(z)}{Q(z)}, \tag{2.38}$$

with $M(\tau)$ defined as

$$M(\tau) = \frac{1}{s!}\prod_{i=1}^{s}(\tau - c_i). \tag{2.39}$$

**Regions of stability:** Using Equation (2.37), it should be clear that the relation between the solution point $x_n$ and the initial value $x_0$ is the following:

$$x_n = (R(\lambda h))^n x_0 = (R(z))^n x_0 \tag{2.40}$$

This is why the region of absolute stability is defined as

$$\{z \in \mathbb{C}; |R(z)| < 1\} \tag{2.41}$$

For these values of $z$, it holds that $x_n \to 0$ as $n \to \infty$. A large region of absolute stability is a desirable property for a method. The relative stability region, which is also called the *Order star*, is the set:

$$\{z \in \mathbb{C}; |R(z)| < |e^z|\} \tag{2.42}$$

The relative stability region compares the growth of the iterations for Dahlquist's equation to the growth of the exact solution $x(t) = x_0 e^{\lambda t}$. Some remarks can be made about a relative stability diagram:

- it is always stable far to the right and always unstable far to the left.
- it shows *finger* shapes around the origin
- a stable finger is in the set of relative stability and contains a zero of $R(z)$
- an unstable finger is not in the set of relative stability and contains a pole of $R(z)$

This will also be clear from the diagrams shown in Section 2.3.3.

**A-stability:** The exact solution of Dahlquist's equation $x(t) = x_0 e^{\lambda t}$ is clearly stable when $\lambda$ is in the left half-plane $\mathbb{C}^-$. A desirable property for an integration method is that it preserves this stability property and this is called A-stability. Therefore a method whose region of absolute stability contains the left half-plane $\mathbb{C}^-$, is called A-stable. Note that the problems described in Section 2.3.1 disappear when an A-stable method is used for a stiff system of equations. The step size $h$ is namely not limited anymore because of stability issues, no matter how large the value of $\max\{|\mathrm{Re}(\lambda_i)|, i = 1, \ldots, m\}$ is. There are also other desirable properties defined for the treatment of stiff systems, such as I-stability (the imaginary axis is stable) and A($\alpha$)-stability. Both of these properties are however weaker than A-stability, which means they are not really important here (see next Section 2.3.3).

**L-stability:** Another desirable stability property that needs to be mentioned is L-stability, which is stronger than A-stability. An A-stable method is at least stable when $z = \lambda h$ lies in the left half-plane $\mathbb{C}^-$. However this does not say everything about the limit when $\lambda h \to -\infty$ and this is the case that can be interesting for stiff systems of equations. If e.g. holds that $|R(\lambda h)| \to 1$ for $\lambda h \to -\infty$, then these stiff

components are damped out only very slowly in comparison to the exact solution. A method is called L-stable if it is A-stable and if the stability function satisfies $R(\lambda h) \to 0$ for $\lambda h \to -\infty$.

### 2.3.3 The Gauss-Legendre methods

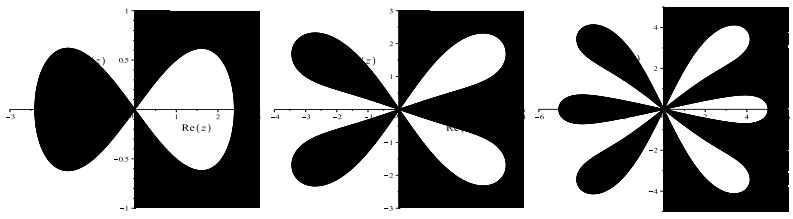
After the previous discussions, it is possible to say something about the stability of the Gauss-Legendre collocation methods for stiff problems. It has already been said that these methods are optimal in terms of order of accuracy, so one might suspect that there is something lost in terms of stability. However, an $s$-stage Gauss method is of order $2s$ and is A-stable which is already a very important property. A proof for this can be found in [33], but the A-stability of the 3 Gauss methods from Section 2.2.1 can also be checked using the definition. The stability functions, corresponding to these 3 methods are the following:

$$R(z) = -\frac{z+2}{z-2}, \qquad s = 1, p = 2 \tag{2.43}$$

$$R(z) = \frac{z^2+6z+12}{z^2-6z+12}, \qquad s = 2, p = 4 \tag{2.44}$$

$$R(z) = -\frac{z^3+12z^2+60z+120}{z^3-12z^2+60z-120}, \quad s = 3, p = 6 \tag{2.45}$$

Note also that the stability function of an $s$-stage Gauss method is the $(s, s)$-Padé approximation for the exponential function $e^z$. The fact that the 3 studied Gauss methods are A-stable means that their region of absolute stability contains the left half-plane $\mathbb{C}^-$. The region of absolute stability of these 3 methods is even exactly equal to only this left half-plane. The regions of relative stability are shown for the 1-, 2- and 3-stage Gauss method respectively in Figure 2.3a, 2.3b and 2.3c. Note that the shaded region is stable in a relative sense, which means $|R(z)| < |e^z|$. The earlier made remarks about relative stability diagrams can clearly be verified in these 3 figures.



(a) 1 stage, order 2 (b) 2 stage, order 4 (c) 3 stage, order 6

Figure 2.3: The relative stability diagrams for the 3 Gauss-Legendre methods.

However, none of these Gauss methods is L-stable, as can be seen from their stability functions in (2.43)-(2.45). They all satisfy $|R(\lambda h)| \to 1$ for $\lambda h \to -\infty$, which

is not a desirable property. This is typical for methods for which the stability region coincides exactly with the negative half-plane as is the case here. The alternative would be for example to use an $s$-stage Radau IIA method which has order $2s-1$ (one order lower than the corresponding Gauss method) but which is L-stable [33]. The Gauss-Legendre methods will be used further in this text. The implementation in the ACADO Toolkit provides more collocation methods and allows to easily add other RK methods to the suite by simply providing their Butcher table.

Note that these methods will be used with completely fixed parameters (step size, number of Newton iterations etc.), because of the real-time constraints for the target applications of the code generation tool. This means that the convergence of the Newton iterations will sometimes be more of a concern than the stability of the RK method. A higher order will therefore be more efficient here, because the corresponding Gauss and Radau IIA methods require a similar amount of work. Again, this is mostly an intuitive decision which will not determine the eventual implementation in ACADO. Finally, it is important to note that the most suitable collocation method also depends on the specific problem.

## 2.4 Solving the nonlinear system

This section will discuss specifically how to solve the system of nonlinear equations from (2.15). These equations can be written in a more standard form $F(x_n, k) = 0$, with $k = (k_1, \ldots, k_s)$:

$$k_i - f(t_n + c_i h, x_n + h \sum_{j=1}^{s} a_{ij} k_j) = 0 \quad \text{for} \quad i = 1, \ldots, s. \tag{2.46}$$

This nonlinear system typically needs to be solved by a Newton method [16]. The full Newton method can be presented as:

**Algorithm 2.1** Newton's method to solve $F(x_n, k) = 0$

**Input:** $k^{[0]}$
**Output:** $k^{[L]}$
  **for** $i = 1 \rightarrow L$ **do**
    $b \leftarrow -F(x_n, k^{[i-1]})$
    $A \leftarrow \frac{\partial F}{\partial k}(x_n, k^{[i-1]})$
    solve $A\Delta k = b$ for $\Delta k$
    $k^{[i]} \leftarrow k^{[i-1]} + \Delta k$
  **end for**

This algorithm is called the full step Newton method, because it evaluates the Jacobian $\frac{\partial F}{\partial k}(x_n, k^{[i-1]})$ in every iteration. Also note that this implementation does a fixed amount of $L$ Newton iterations. This is obviously not the best implementation to make sure the method converges, but it is the only implementation possible for real-time applications. Since everything here is implemented in the code generation tool,

this real-time view on the discussed algorithms is very important. The alternative would be to use a stopping criterium, e.g. based on the norm of the correction value $\|\Delta k\|$. The most time consuming steps in Algorithm 2.1 clearly are the evaluation of the Jacobian and the solution of the linear system $A\Delta k = b$. Both of these steps will be discussed in more detail.

### 2.4.1 Evaluation of the Jacobian

Remember that the function $F(x_n, k)$ is defined as

$$F(x_n, k) = \begin{pmatrix} k_1 - f(t_n + c_1 h, x_n + h\sum_{j=1}^{s} a_{1j} k_j) \\ k_2 - f(t_n + c_2 h, x_n + h\sum_{j=1}^{s} a_{2j} k_j) \\ \vdots \\ k_s - f(t_n + c_s h, x_n + h\sum_{j=1}^{s} a_{sj} k_j) \end{pmatrix} \tag{2.47}$$

This means that the Jacobian $\frac{\partial F}{\partial k}$ is an $sN_x \times sN_x$ matrix that can be evaluated as follows:

$$\frac{\partial F}{\partial k}(x_n, k) = \begin{pmatrix} \mathbb{1} - ha_{11}\frac{\partial f}{\partial x}(t_n + c_1 h, \ldots) & \cdots & -ha_{1s}\frac{\partial f}{\partial x}(t_n + c_1 h, \ldots) \\ \vdots & \ddots & \vdots \\ -ha_{s1}\frac{\partial f}{\partial x}(t_n + c_s h, \ldots) & \cdots & \mathbb{1} - ha_{ss}\frac{\partial f}{\partial x}(t_n + c_s h, \ldots) \end{pmatrix}, \tag{2.48}$$

where the Jacobian $\partial f/\partial x$ is an $N_x \times N_x$ matrix and $\mathbb{1}$ is the $N_x \times N_x$ identity matrix. The evaluation of the complete Jacobian needs the step size $h$, the internal coefficients $a_{ij}$ and $s$ different evaluations of the Jacobian $\partial f/\partial x$:

$$\frac{\partial f}{\partial x}(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij} k_j) \quad \text{for} \quad i = 1, 2, \ldots, s. \tag{2.49}$$

These evaluations of the Jacobian $\partial f/\partial x$ can preferably be done using Algorithmic Differentiation (AD), which is supported by the ACADO Toolkit. A good description of AD can be found in [16], but a more detailed and authoritative textbook on AD is [31].

**Algorithmic Differentiation (AD):** Each differentiable function consists of several elementary operations such as multiplication, addition, sine-functions etc. The principle of AD requires that the function is available in the form of source code in a standard programming language such as C/C++. It is then possible for an AD-tool to process this code to generate new code that delivers the desired derivatives. Let us assume that the function $f : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$ is composed of a sequence of $m$ elementary operations $\phi_i$, $i = 0, 1, \ldots, m-1$. The idea of AD is then to use the chain rule and differentiate each of the elementary operations $\phi_i$ separately [16]. For the generation of first order derivatives, there is the *forward* and the *backward* mode of AD.

The forward mode of AD is slightly more expensive than using numerical finite differences, but it is exact up to machine precision while finite differences always lead

to a loss of precision of half the valid digits. The backward mode of AD can be much faster than even using finite differences for a function $h : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ if $n_2 \ll n_1$. But for the function $f : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$, the forward mode is preferable. In this case, the forward mode is faster and the backward mode of AD needs to store all intermediate variables and partial derivatives. More information on available AD implementations can be found in [2, 31], but it is also available in the ACADO Toolkit.

### 2.4.2 Solving the linear system

After the evaluation of the Jacobian, the next step of Algorithm 2.1 which needs to be discussed is the solution of the linear system. There are many possible algorithms which can be used to solve this linear system. First it is important to note that a direct method is preferable here instead of an iterative method to solve the linear system. A direct method needs typically $O(m^3)$ operations to solve a linear system involving a $m \times m$ matrix, while iterative methods can improve the total work from $O(m^3)$ to $O(m^2)$ [57]. The use of iterative methods, however, is mostly recommended for very large systems and ACADO is currently not designed for large-scale systems anyway [35]. Most systems in this toolkit have 10-100 states, which means that the linear system in Algorithm 2.1 will involve $20 - 200$ equations in case of a 2-stage RK method. Also, the performance of iterative methods strongly depends on the condition number of the linear system, which for stiff problems can be very high.

A more important reason to prefer a direct method in this situation is again because of the real-time applications of the code generation tool. An implementation of an iterative method for code generation will namely need to do a fixed amount of iterations. This number of iterations depends on the desired accuracy for the solution and on the quality of the first guess. It is however not evident how to get a reasonable first guess for the solution of the system $A\Delta k = b$ and especially how to guarantee a certain quality of this first guess. So this would mean that a large number of iterations would need to be used, mostly eliminating the advantages of using an iterative method. A final reason why a direct method will be used in this project, concerns the reuse of existing factorizations and is described in the next Section 2.4.3.

Note that it is not one of the goals of this thesis to find out which of the direct methods to solve a linear system is the best one. The implementation namely allows one to easily replace the linear solver by a different one. There are two standard ways to solve dense linear systems: using the QR factorization or using the LU factorization of the matrix. A principal method for computing QR factorizations is the Householder triangularization. This algorithm requires $\sim \frac{4}{3}m^3$ flops for an $m \times m$ matrix. Gaussian elimination is however *the* standard method to solve linear systems and uses the LU factorization of the matrix. The work needed for the Gaussian elimination is $\sim \frac{2}{3}m^3$ flops, which is half the amount of work needed for the Householder triangularization for sufficiently large values of $m$. This is the reason why Gaussian elimination will be used here.

Note however that Gaussian elimination without pivoting is unstable [57]. Gaussian elimination with complete pivoting (interchanging rows or columns) can be shown to be stable [29] but then the total cost of selecting pivots becomes $O(m^3)$. That is why complete pivoting is almost never used in practice. The alternative way to make this algorithm more stable is by using partial pivoting. In this case only rows can be interchanged and the total cost for selecting the pivot at each step becomes only $O(m^2)$. For Gaussian elimination with partial pivoting there exist examples to show that it is not stable. These examples are however quite contrived and experience over many years has shown that Gaussian elimination with partial pivoting is stable for most problems occurring in practice [29]. This partial pivoting namely ensures that no multiplier is larger than one in absolute value, so it effectively guards against arbitrarily large multipliers. There appears to be no practical justification for choosing complete pivoting over partial pivoting except in cases where rank determination is an issue [30]. The conclusion of all this is that Gaussian elimination with partial pivoting is the method of choice here. Efficient implementations of the mentioned algorithms can be found in [57].

Note that the complete LU factorization of a matrix $A$ can be stored in the amount of memory which was used to store the matrix $A$, which is an important fact for code generation. The reason for this is that $L$ is a lower-triangular and $U$ an upper-triangular matrix and all the diagonal values of $L$ are equal to 1, so they do not need to be saved. This leads to the following efficient implementation of the Gaussian elimination with partial pivoting to compute the LU factorization of a $m \times m$ matrix:

**Algorithm 2.2** Gaussian Elimination with Partial Pivoting [57]

**Input:** $A$, $b$
**Output:** $T$ containing the LU factorization, $P$, $y$

```
T ← A (renaming, same memory)
y ← b (also renaming)
P ← (1, 2, ..., m) (initialization vector P)
for i = 1 → m − 1 do
    Select r ≥ i to maximize |T(r, i)|
    T(i, :) ↔ T(r, :) (interchange rows)
    P(i) ↔ P(r)
    y(i) ↔ y(r)
    for j = i + 1 → m do
        T(j, i) ← T(j, i)/T(i, i)
        T(j, i + 1 : m) ← T(j, i + 1 : m) − T(j, i)T(i, i + 1 : m)
        y(j) ← y(j) − T(j, i)y(i)
    end for
end for
```

Note that the output vector $P$ of this Algorithm 2.2 contains the necessary information about the row swaps which were done because of the pivot selections.

The purpose of this output will be clear when looking at Algorithm 2.5, described in the next section. Algorithm 2.2 also applies all the transformations to the vector $b$, which is the right-hand side of the linear system. The result of this is that the outputs $T$ and $y$ of this algorithm can be forwarded directly to Algorithm 2.3 which performs a back substitution. By also applying all the transformations to the vector $b$, the resulting vector $y$ namely satisfies $Ly = b$. This means that the solution $\Delta k$ of the linear system $A\Delta k = b$ can be found by solving the upper-triangular system $U\Delta k = y$ which is typically done using back substitution. This algorithm requires only $\sim m^2$ flops, which is negligible with respect to the amount of $\sim \frac{2}{3}m^3$ flops needed for Algorithm 2.2.

**Algorithm 2.3** Back Substitution

**Input:** $T$, $y$ from Algorithm 2.2
**Output:** $\Delta k \Rightarrow A\Delta k = b$
  **for** $i = m \rightarrow 1$ **do**
    $\Delta k(i) \leftarrow y(i)$
    **for** $j = i + 1 \rightarrow m$ **do**
      $\Delta k(i) \leftarrow \Delta k(i) - T(i, j)\Delta k(j)$
    **end for**
    $\Delta k(i) \leftarrow \Delta k(i)/T(i, i)$
  **end for**

### 2.4.3 Reuse of a factorization

As already mentioned, the Newton method in Algorithm 2.1 is the full Newton method. This means that an evaluation and a factorization of the Jacobian $\frac{\partial F}{\partial k}$ is done in every iteration. If the first guess $k^{[0]}$ for the variables $k = (k_1, \ldots, k_s)$ is sufficiently good, then a few Newton iterations will be sufficient and the successive values of the Jacobian will not differ much. So, assuming the first guess $k^{[0]}$ is indeed sufficiently good, Algorithm 2.4 is much more efficient than Algorithm 2.1. In what follows, this assumption about $k^{[0]}$ will always be made and Algorithm 2.4 will always be used. The result is that the initialization of the variables $k$, discussed in the next section, becomes even more important.

**Algorithm 2.4** Modified Newton method to solve $F(x_n, k) = 0$

**Input:** $k^{[0]}$
**Output:** $k^{[L]}$
  $b \leftarrow -F(x_n, k^{[0]})$
  $A \leftarrow \frac{\partial F}{\partial k}(x_n, k^{[0]})$
  solve $A\Delta k = b$ for $\Delta k$ (using Algorithm 2.2 + 2.3)
  $k^{[1]} \leftarrow k^{[0]} + \Delta k$
  **for** $i = 2 \rightarrow L$ **do**
    $b \leftarrow -F(x_n, k^{[i-1]})$
    solve $A\Delta k = b$ for $\Delta k$ (using Algorithm 2.5)
    $k^{[i]} \leftarrow k^{[i-1]} + \Delta k$
  **end for**

The algorithm to solve the linear system $A\Delta k = b$, reusing the previously computed LU factorization of the matrix $A$, is described by Algorithm 2.5. Note that this algorithm only requires $\sim 2m^2$ flops instead of the $\sim \frac{2}{3}m^3$ flops of Algorithm 2.2. That is the reason why the Newton method from Algorithm 2.4, which reuses the first factorization of the Jacobian in the subsequent iterations, is so efficient.

**Algorithm 2.5** Reuse of the LU factorization from Algorithm 2.2

**Input:** $T$, $P$ from Algorithm 2.2 and $b$
**Output:** $\Delta k$
  **for** $i = 1 \rightarrow m$ **do**
    $y(i) \leftarrow b(P(i))$
  **end for**
  **for** $i = 1 \rightarrow m - 1$ **do**
    **for** $j = i + 1 \rightarrow m$ **do**
      $y(j) \leftarrow y(j) - T(j, i)y(i)$
    **end for**
  **end for**
  **for** $i = m \rightarrow 1$ **do**
    $\Delta k(i) \leftarrow y(i)$
    **for** $j = i + 1 \rightarrow m$ **do**
      $\Delta k(i) \leftarrow \Delta k(i) - T(i, j)\Delta k(j)$
    **end for**
    $\Delta k(i) \leftarrow \Delta k(i)/T(i, i)$
  **end for**

## 2.5 Initialization of the Variables

To be able to do only a few Newton iterations per integration step, it is important to have a good initial guess $k^{[0]}$ for the variables $k$ in Algorithm 2.4. This section will discuss this initialization of the variables. A distinction will be made between

the initialization of the first step and the initialization of a subsequent step, which means the information of the previous step can be used.

### 2.5.1 Initialization of the First Step

It is important to discuss the initialization of the first step separately because of two reasons. The first reason is that there is no information from a previous integration step available in this case. The second reason is that errors accumulate over the integration interval, so the error made in the first step will influence the quality of all the following results. This justifies potentially extra effort being made in the initialization of the first step. In addition, if only some extra computations are done in the first step to get a better initialization of the variables $k$ then this will not be a huge increase in the total time to integrate the system over a certain interval.

This also means that it does not matter too much how this initialization of the variables in the first step is performed, as long as it is efficient in the sense that it generally improves the performance of the integrator. The variables could for example be initialized using an explicit (RK) method. It is, however, easier to just do one or more extra Newton iterations in Algorithm 2.4 in the case of this first integration step. The best choice for this number of extra Newton iterations will be studied in Section 6.5.1. This means that the initialization value $k^{[0]}$ for this first integration step will simply be equal to zero.

### 2.5.2 Initialization of Subsequent Steps

The situation for the initialization of the variables $k$ in a subsequent step is a bit different, because now the information from the previous integration step is available. Two different possibilities for this initialization will be studied here and compared numerically in Section 6.5.2.

**Warm-start:** The first possibility is to just use the converged values for the variables $k$ from the previous step, as the initial guess for this next step. In the implementation, this requires no work because the same memory for the variables $k$ is still used. It is therefore even easier than setting the variables to zero, an initialization which is generally worse. This initialization will work well in the case of a small step size $h$ and a slowly varying function $f(t, x(t))$.

**Extrapolation:** The second possibility is to really calculate a first estimate for the values $f(t_n + (1 + c_i)h)$, with $i = 1, \ldots, s$, given the converged values for the variables $k$ from the previous step. This again could be done using an explicit (RK) method. It is however more efficient to just make use of the continuous interpolating polynomial of the collocation method. It is namely possible to extrapolate the value of the polynomial $\dot{p}(t)$ to the quadrature nodes $c_i$ on the next integration interval. This idea has for example been described in [11]. The initialization values are then given by the following equation (using Equation (2.17)):

$$\dot{p}(t_n + h + c_i h) = \sum_{j=1}^{s} k_j l_j (1 + c_i),\ i = 1, \ldots, s \tag{2.50}$$

It is clear that for this method the following $s \times s$ matrix $D$ is needed when an $s$-stage collocation method is used:

$$D = \left(d_{ij}\right) = \begin{pmatrix} l_1(1+c_1) & l_2(1+c_1) & \cdots & l_s(1+c_1) \\ l_1(1+c_2) & l_2(1+c_2) & \cdots & l_s(1+c_2) \\ \vdots & \ddots & & \vdots \\ l_1(1+c_s) & l_2(1+c_s) & \cdots & l_s(1+c_s) \end{pmatrix} \tag{2.51}$$

These matrices for the 1-, 2- and 3-stage Gauss-Legendre collocation method are stated respectively in (2.52), (2.53) and (2.54). Note that it is clear from (2.52) that both described initializations are the same for the 1-stage Gauss method.

$$\begin{pmatrix} 1 \end{pmatrix} \tag{2.52}$$

$$\begin{pmatrix} (3+\sqrt{3})/\sqrt{3} & -\sqrt{3} \\ \sqrt{3} & (-3+\sqrt{3})/\sqrt{3} \end{pmatrix} \tag{2.53}$$

$$\begin{pmatrix} 1/15(-10+\sqrt{15})(-5+\sqrt{15}) & -20/3+4/3\sqrt{15} & 10/3-1/3\sqrt{15} \\ 10/3-1/3\sqrt{15} & -17/3 & 10/3+1/3\sqrt{15} \\ 10/3+1/3\sqrt{15} & -20/3-4/3\sqrt{15} & 1/15(10+\sqrt{15})(5+\sqrt{15}) \end{pmatrix} \tag{2.54}$$

## 2.6 Conclusion

The collocation methods form an important subset of the large set of implicit RK methods. The focus was mainly on the Gauss-Legendre methods because they are A-stable and optimal in terms of their order of accuracy, but other collocation methods exist with useful properties. What makes the collocation methods so attractive is that they can provide a continuous approximation of the solution over the integration interval at a relatively low cost. Taking into account the initialization of the variables, code generation for these methods can be implemented quite efficiently using Algorithmic Differentiation and an exported linear solver. Each evaluation of the Jacobian and its computed factorization will namely be reused as much as possible. The continuous output of the collocation methods has also proven to be useful in the initialization of the variables.

# Chapter 3

# Derivative Generation

Some online algorithms to numerically solve estimation problems and optimal control problems were presented roughly in Chapter 1. From this discussion, it is clear why the sensitivities of the solution of the IVP (Equation (2.2)) with respect to the unknowns are required. In this chapter the generation of these derivatives of the solution with respect to the previous solution point $x_n$, the control inputs $u_n$ and optionally some parameters $p$ will be discussed. Note that the whole integration over one shooting interval is treated as one function. In the case of multiple integration steps per shooting interval, the sensitivities are updated as follows

$$\frac{\partial x(t_{n+1})}{\partial x_0} = \frac{\partial x(t_{n+1})}{\partial x_n}\frac{\partial x(t_n)}{\partial x_{n-1}}\cdots\frac{\partial x(t_1)}{\partial x_0} \tag{3.1}$$

$$\begin{cases} \dfrac{\partial x(t_{n+1})}{\partial u} = \dfrac{\partial x(t_{n+1})}{\partial x_n}\dfrac{\partial x(t_n)}{\partial u} + \dfrac{\partial x(t_{n+1})}{\partial u} \\ \dfrac{\partial x(t_n)}{\partial u} = \dfrac{\partial x(t_n)}{\partial x_{n-1}}\dfrac{\partial x(t_{n-1})}{\partial u} + \dfrac{\partial x(t_n)}{\partial u} \\ \qquad\qquad \cdots \end{cases} \tag{3.2}$$

It is also important to further clarify which sensitivities are wanted because there are actually three possibilities. It is namely possible to be talking about the sensitivities of

- the exact true simulation results (not available)
- the exact solutions of the discretized system (solved to high precision, so also not available)
- or the numerical results for the discretized system, using only a few Newton iterations

The numerical results for the discretized system are the ones that are used in the optimization algorithm, so their derivatives are needed here. This will strongly influence the way these derivatives will be computed.

Let us assume in what follows that $w_n = (x_n, u_n, p)$ denotes all the variables with respect to which the derivatives of the results are needed. This means that different approaches to calculate the $N_x \times (N_x + N_u + N_p)$ matrix $\frac{\partial x(t_{n+1})}{\partial w_n}$ will be described and evaluated in this section. The collocation methods from the previous chapter (described by Equation (2.15)) will be presented here by the following equations:

$$\begin{aligned} F(w_n, k) &= 0, \\ x_{n+1} &= \phi(x_n, k), \end{aligned} \tag{3.3}$$

with $k = (k_1, \ldots, k_s)$ from the discussion in Section 2.2. The function $\phi(x_n, k)$ determines the explicit update formula of the collocation method. Similarly, the Newton iterations from Algorithm 2.4 to solve the nonlinear system $F(w_n, k) = 0$ will be presented in this chapter by the following equations:

$$\begin{aligned} M &= \frac{\partial F}{\partial k}(w_n, k^{[0]}) \\ k^{[i]} &= k^{[i-1]} - M^{-1} F(w_n, k^{[i-1]}), \;\; i = 1, \ldots, L \end{aligned} \tag{3.4}$$

Section 3.1 presents a method based on the Implicit Function Theorem (IFT), the principle of Internal Numerical Differentiation (IND) is discussed in Section 3.2 and the Variational Differential Equations (VDE) are treated in Section 3.3. While this chapter will concentrate on forward methods, the possibility of adjoint sensitivity generation is discussed briefly in Section 3.4.

## 3.1 Implicit Function Theorem (IFT)

The first approach to calculate the derivatives in $\frac{\partial x(t_{n+1})}{\partial w_n}$ is obtained by applying the Implicit Function Theorem (IFT) to (3.3). This is equivalent to first differentiating equation $F(w_n, k) = 0$ with respect to $w_n$ like this:

$$\begin{aligned} &\frac{\partial F}{\partial w_n}(w_n, k) + \frac{\partial F}{\partial k}(w_n, k)\frac{dk}{dw_n} = 0 \\ &\qquad\qquad\qquad\qquad \Downarrow \\ &\qquad\qquad\qquad\qquad \frac{dk}{dw_n} = -\left(\frac{\partial F}{\partial k}(w_n, k)\right)^{-1} \frac{\partial F}{\partial w_n}(w_n, k) \end{aligned} \tag{3.5}$$

After differentiating both sides of equation $x_{n+1} = \phi(x_n, k)$, the previous result can be used like this:

$$\begin{aligned} &\frac{\partial x_{n+1}}{\partial w_n} = \frac{\partial \phi}{\partial x_n}\frac{dx_n}{dw_n} + \frac{\partial \phi}{\partial k}\frac{dk}{dw_n} \\ &\qquad \Downarrow \\ &\frac{\partial x_{n+1}}{\partial w_n} = \frac{\partial \phi}{\partial x_n}\frac{dx_n}{dw_n} - \frac{\partial \phi}{\partial k}\left(\frac{\partial F}{\partial k}(w_n, k)\right)^{-1} \frac{\partial F}{\partial w_n}(w_n, k) \end{aligned} \tag{3.6}$$

Remember that $x_{n+1} = \phi(x_n, k)$ is the compact version of the update formula $x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i$ of the used collocation method, so all the variables appear

linearly in this function $\phi(x_n, k)$. This means that these derivatives of the function $\phi$ will just be some constants that need to be added or multiplied. The dominating computations in this equation will therefore be the evaluations of $\frac{\partial F}{\partial w_n}(w_n, k)$ and $\frac{\partial F}{\partial k}(w_n, k)$ and the computation of $(\frac{\partial F}{\partial k}(w_n, k))^{-1} \frac{\partial F}{\partial w_n}(w_n, k)$. This last computation is done by solving the corresponding linear system instead of explicitly calculating the inverse of the matrix. For the evaluations, the variables $k$ are set to the converged values $k^{[L]}$ after the $L$ Newton iterations. So eventually the IFT method to generate the sensitivities $\frac{\partial x(t_{n+1})}{\partial w_n}$ looks like this:

$$\begin{aligned} \frac{\partial F}{\partial k}(w_n, k^{[L]}) \frac{dk}{dw_n} = -\frac{\partial F}{\partial w_n}(w_n, k^{[L]}) \Rightarrow \frac{dk}{dw_n} \\ \frac{\partial x_{n+1}}{\partial w_n} = \frac{\partial \phi}{\partial x_n} \frac{dx_n}{dw_n} + \frac{\partial \phi}{\partial k} \frac{dk}{dw_n} \end{aligned} \tag{3.7}$$

In the implementation of this IFT method, every column of the $N_x \times (N_x + N_u + N_p)$ matrix $\frac{\partial x_{n+1}}{\partial w_n}$ is calculated separately. This means that only for the first column, the matrix $\frac{\partial F}{\partial k}(w_n, k^{[L]})$ needs to be evaluated and the LU factorization needs to be computed to solve the linear system (Section 2.4.2). For all the subsequent columns, this matrix evaluation and LU factorization can be reused by using Algorithm 2.5 which needs only $\sim 2m^2$ flops instead of the $\sim \frac{2}{3}m^3$ flops of Algorithm 2.2. Note that for every column of $\frac{\partial x_{n+1}}{\partial w_n}$, a different column of the matrix $\frac{\partial F}{\partial w_n}(w_n, k^{[L]})$ needs to be used. As a final remark, the evaluations of $\frac{\partial F}{\partial w_n}$ and $\frac{\partial F}{\partial k}$ can be worked out similarly as in Section 2.4.1. This means that both matrix evaluations eventually make use of the same Jacobian $\frac{\partial f}{\partial w_n}$ which can be evaluated using Algorithmic Differentiation (AD).

### 3.1.1 An efficient alternative implementation (IFT-R)

From the previous discussion, it should be clear how (3.7) can be used to generate the derivatives $\frac{\partial x(t_{n+1})}{\partial w_n}$ after using Algorithm 2.4 in combination with the update formula from (2.15) to compute the next solution point $x_{n+1}$. This would mean that two LU factorizations of a $sN_x \times sN_x$ matrix are needed in every integration step. There is however an alternative implementation possible which is much more efficient. In the implementation of the Newton method of Algorithm 2.4, a matrix evaluation and factorization is done for the first Newton iteration. When using the IFT method to generate the sensitivities, there is still a LU factorization of the matrix $\frac{\partial F}{\partial k}(w_n, k^{[L]})$ from the previous integration step available. This matrix can serve as a decent approximation for the Jacobian needed in the Newton method of the next integration step. Chapter 6 will present numerical experiments which confirm this. The important result is that this alternative implementation only needs one LU factorization of a $sN_x \times sN_x$ matrix in every integration step, except for the first step in which still two factorizations are needed. In the first step, there is namely no previous step from which a LU factorization can be reused. It should be clear that this alternative implementation can be much more efficient because the

LU factorizations (Algorithm 2.2) will be the dominating cost in the integrator. This is further referred to as the IFT-R method (IFT with reuse of the Jacobian).

## 3.2 Internal Numerical Differentiation (IND)

The second approach is based on the fundamental principle of Internal Numerical Differentiation (IND) due to Bock [6]. The idea is to generate the needed derivatives by differentiation of the integrator method itself. The major difference with External Numerical Differentiation (END) is that for IND the adaptive parts of the method (such as error control and step size selection as well as matrix factorizations) get frozen during sensitivity generation. The adaptive parts are therefore only evaluated for the nominal solution [38]. In this text, the implementations of all the methods are however designed for real-time applications which means that adaptive parts are avoided. The only difference left between IND and END is that matrix factorizations are reused for sensitivity generation using IND.

### 3.2.1 Finite Differences (FD)

The easiest way to implement IND is by using Finite Differences (FD) like this:

$$\frac{\partial x(t_{n+1}; w_n)}{\partial w_n}\lambda \approx \frac{x(t_{n+1}; w_n + \varepsilon\lambda) - x(t_{n+1}; w_n)}{\varepsilon} \tag{3.8}$$

The principle of IND is then to integrate the nominal and the disturbed trajectories (respectively denoted by $x(t_{n+1}; w_n)$ and $x(t_{n+1}; w_n + \varepsilon\lambda)$) using the exact same discretization scheme. This way, the integration method itself will indeed be differentiated. Note that typically the value $\sqrt{\text{eps}}$ is being used for $\varepsilon$ where eps is the machine precision, because this value of $\sqrt{\text{eps}}$ is small enough without producing a large rounding error. But one can still only expect an accuracy of roughly the square-root of this machine precision. For the disturbed trajectories, the Newton iterations from (3.4) are performed reusing the Jacobian $M$ and its factorization from the nominal trajectory. Using the principle of IND, the adaptive parts of the method are namely frozen during sensitivity generation.

### 3.2.2 Algorithmic Differentiation (IND-AD)

The same principle of IND, previously implemented using FD, can also be implemented using Algorithmic Differentiation (AD). The principle of AD itself has already been introduced briefly in Section 2.4.1. First, remember that (3.4) presented the Newton iterations from Algorithm 2.4 to solve the nonlinear system $F(w_n, k) = 0$. To get the complete description of the collocation method, the initialization of the variables $k$ from Section 2.5.2 and the explicit update formula need to be added:

$$\begin{aligned} k^{[0]} &= h(w_n, k_{\text{prev}}) \\ k^{[i]} &= k^{[i-1]} - M^{-1}F(w_n, k^{[i-1]}),\ i = 1, \ldots, L \\ x_{n+1} &= \phi(x_n, k^{[L]}) \end{aligned} \tag{3.9}$$

The function $h$ can depend on $w_n = (x_n, u_n, p)$ if e.g. an explicit method is being used for the initialization of the variables $k$. And it can also depend on the converged values $k_{\text{prev}}$ of the variables of the previous integration step. This is the case when these values $k_{\text{prev}}$ are used as an initial guess or when the continuous output of the collocation method is extrapolated to achieve an initialization (Section 2.5.2). The idea here is to directly differentiate the equations from (3.9) with respect to $w_n$ which results in

$$\begin{aligned}
\frac{dk^{[0]}}{dw_n} &= \frac{\partial h(w_n, k_{\text{prev}})}{\partial w_n} \\
\frac{dk^{[i]}}{dw_n} &= \frac{dk^{[i-1]}}{dw_n} - M^{-1}\left(\frac{\partial F(w_n, k^{[i-1]})}{\partial w_n} + \frac{\partial F(w_n, k^{[i-1]})}{\partial k}\frac{dk^{[i-1]}}{dw_n}\right), \quad i = 1, \dots, L \\
\frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial \phi(x_n, k^{[L]})}{\partial x_n}\frac{dx_n}{dw_n} + \frac{\partial \phi(x_n, k^{[L]})}{\partial k}\frac{dk^{[L]}}{dw_n}
\end{aligned} \tag{3.10}$$

This is clearly another iterative scheme which can be used to generate the sensitivities $\frac{\partial x(t_{n+1})}{\partial w_n}$. In what follows, this method to generate the sensitivities will be called the IND-AD method. The matrix evaluations in these equations make namely use of the Jacobian $\frac{\partial f}{\partial w_n}$ which is evaluated using AD. Note that the same matrix $M$ is used here as the one in (3.9) which means that no LU factorizations need to be computed for the IND-AD method. The linear systems (multiplications with $M^{-1}$) can just be solved using Algorithm 2.5. The disadvantage of this method to generate the derivatives is however that every iteration needs the evaluation of

$$\frac{\partial F(w_n, k^{[i-1]})}{\partial w_n} + \frac{\partial F(w_n, k^{[i-1]})}{\partial k}\frac{dk^{[i-1]}}{dw_n} \tag{3.11}$$

This can be partly avoided by generating the derivatives after the iterations of (3.9), which means that only the converged values $k^{[L]}$ will be used like this:

$$\begin{aligned}
\frac{dk^{[0]}}{dw_n} &= \frac{\partial h(w_n, k_{\text{prev}})}{\partial w_n} \\
\frac{dk^{[i]}}{dw_n} &= \frac{dk^{[i-1]}}{dw_n} - M^{-1}\left(\frac{\partial F(w_n, k^{[L]})}{\partial w_n} + \frac{\partial F(w_n, k^{[L]})}{\partial k}\frac{dk^{[i-1]}}{dw_n}\right), \quad i = 1, \dots, L \\
\frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial \phi(x_n, k^{[L]})}{\partial x_n}\frac{dx_n}{dw_n} + \frac{\partial \phi(x_n, k^{[L]})}{\partial k}\frac{dk^{[L]}}{dw_n}
\end{aligned} \tag{3.12}$$

This implementation clearly is more efficient and will therefore be used. To better understand how this iterative scheme is able to generate the correct derivatives, the convergence will now be discussed.

**Convergence to the results of the IFT method:** The equations of the IND-AD method in (3.12) to generate the derivatives converge to the same results as

those obtained using the IFT method (Section 3.1). An intuitive mathematical proof will be given here to gain more insight in these methods.

The first observation is that the last equation of (3.12) is already the same as for the IFT method in (3.7). So the fact that needs to be proven is

$$\frac{dk^{[i]}}{dw_n} \to \frac{dk}{dw_n} = -\left(\frac{\partial F(w_n, k^{[L]})}{\partial k}\right)^{-1} \frac{\partial F(w_n, k^{[L]})}{\partial w_n} \text{ for } i \to \infty \tag{3.13}$$

From Equation (3.12), the following is however already known:

$$\begin{aligned} \frac{dk^{[0]}}{dw_n} &= \frac{\partial h(w_n, k_{\text{prev}})}{\partial w_n} \\ \frac{dk^{[i]}}{dw_n} &= \frac{dk^{[i-1]}}{dw_n} - M^{-1}\left(\frac{\partial F(w_n, k^{[L]})}{\partial w_n} + \frac{\partial F(w_n, k^{[L]})}{\partial k}\frac{dk^{[i-1]}}{dw_n}\right) \\ &= \left(\mathbb{1} - M^{-1}\frac{\partial F(w_n, k^{[L]})}{\partial k}\right)\frac{dk^{[i-1]}}{dw_n} - M^{-1}\frac{\partial F(w_n, k^{[L]})}{\partial w_n} \end{aligned} \tag{3.14}$$

Substituting the first equation into the second one gives:

$$\begin{aligned} \frac{dk^{[0]}}{dw_n} &= \frac{\partial h(w_n, k_{\text{prev}})}{\partial w_n} \Rightarrow \\ \frac{dk^{[i]}}{dw_n} &= \left(\mathbb{1} - M^{-1}\frac{\partial F(w_n, k^{[L]})}{\partial k}\right)^{i} \frac{\partial h(w_n, k_{\text{prev}})}{\partial w_n} \\ &\quad - \sum_{j=0}^{i-1}\left(\mathbb{1} - M^{-1}\frac{\partial F(w_n, k^{[L]})}{\partial k}\right)^{j} M^{-1}\frac{\partial F(w_n, k^{[L]})}{\partial w_n} \end{aligned} \tag{3.15}$$

Using a simplified notation, this can be written as:

$$\frac{dk^{[i]}}{dw_n} = \left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^{i} \frac{\partial h}{\partial w_n} - \sum_{j=0}^{i-1}\left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^{j} M^{-1}\frac{\partial F}{\partial w_n} \tag{3.16}$$

Remember that the matrix $M$ is an approximation for the Jacobian $\frac{\partial F(w_n, k^{[L]})}{\partial k}$. This means that $M \approx \frac{\partial F}{\partial k}$ or $M^{-1}\frac{\partial F}{\partial k} \approx \mathbb{1}$. The accuracy of this approximation depends on the initialization of the variables $k$ with the values $k^{[0]} = h(w_n, k_{\text{prev}})$. This initialization needs to be sufficiently good, so that $|\lambda_i| < 1$ holds for every eigenvalue $\lambda_i$ of the matrix $\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}$. This will namely ensure the convergence of the powers of this matrix to the zero matrix:

$$\left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^{n} \to \mathbb{0} \text{ for } n \to \infty \tag{3.17}$$

If the matrix $M$ is a sufficiently good approximation for the Jacobian $\frac{\partial F}{\partial k}$, Equation (3.16) becomes:

$$\begin{aligned}\frac{dk^{[i]}}{dw_n} &= -\sum_{j=0}^{i-1}\left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^j M^{-1}\frac{\partial F}{\partial w_n} \text{ for } i \to \infty \\ &= -\sum_{j=0}^{\infty}\left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^j M^{-1}\frac{\partial F}{\partial w_n}\end{aligned} \tag{3.18}$$

Note that because the condition $|\lambda_i| < 1$ is met for every eigenvalue $\lambda_i$ of the matrix $\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}$, also the following holds:

$$\begin{aligned}\sum_{j=0}^{\infty}\left(\mathbb{1} - M^{-1}\frac{\partial F}{\partial k}\right)^j &= \left(\mathbb{1} - (\mathbb{1} - M^{-1}\frac{\partial F}{\partial k})\right)^{-1} \\ &= \left(M^{-1}\frac{\partial F}{\partial k}\right)^{-1} \\ &= \frac{\partial F}{\partial k}^{-1} M\end{aligned} \tag{3.19}$$

Substituting this in (3.18) finally results in the following equation for the derivatives of the variables $k$:

$$\frac{dk^{[\infty]}}{dw_n} = -\frac{\partial F}{\partial k}^{-1}\frac{\partial F}{\partial w_n} \tag{3.20}$$

This last result is exactly what is stated in (3.13), which needed to be proven. This clearly demonstrates that the iterative scheme of (3.12) generates the correct derivatives if sufficient iterations are performed. Note that the assumption that $M$ is a sufficiently good approximation for the Jacobian $\frac{\partial F}{\partial k}$ is the same as for the convergence of the Newton iterations for the next solution point $x_{n+1}$. The major disadvantage of the IND-AD method is that it is costly when computing many directional derivatives, as is done here.

## 3.3 Variational Differential Equations (VDE)

For this last approach, the sensitivity matrix $S_w(t;t_n)$ is explicitly used to denote the derivatives which need to be generated.

$$S_w(t;t_n) = \frac{\partial x(t;t_n,w_n)}{\partial w_n} \tag{3.21}$$

Remember that the most general form of the IVP that needs to be solved, is the following

$$\begin{aligned}\dot{x}(t) &= f(t,x(t),u(t),p) \\ x(t_n) &= x_n\end{aligned} \tag{3.22}$$

The Variational Differential Equations (VDE) can then be obtained by differentiating this IVP formulation with respect to $w_n$:

$$\begin{aligned} \frac{\partial S_w(t;t_n)}{\partial t} &= \frac{\partial f(t,x(t),u(t),p)}{\partial x} S_w(t;t_n) + \frac{\partial f(t,x(t),u(t),p)}{\partial w_n}, \\ S_w(t_n;t_n) &= \frac{\partial x(t_n;t_n,w_n)}{\partial w_n}, \end{aligned} \tag{3.23}$$

with Jacobian $\frac{\partial f}{\partial w_n} = [\,\mathbb{0}\,|\,\frac{\partial f}{\partial u}\,|\,\frac{\partial f}{\partial p}\,]$ which is evaluated using AD and $\frac{\partial x}{\partial w_n} = [\,\mathbb{1}\,|\,\mathbb{0}\,|\,\mathbb{0}\,]$. The principle of this approach is to augment the ODE system (Equation (3.22)) by these differential equations whose unique solution is the needed sensitivity matrix $S_w(t;t_n)$ [38]. This method is however more suitable for an explicit instead of an implicit integration method. For an explicit integration method, the amount of work namely is simply proportional to the number of equations. Assuming however that the LU factorization is the dominant cost for an implicit integration method, the amount of work is here proportional to the third power of this number of equations. This explains why augmenting the ODE system with extra equations would be too expensive for an implicit integrator. It is possible to exploit the structure that is present in this augmented system, but that would lead to a method which is more or less equivalent to the other methods to generate the sensitivities.

## 3.4 Adjoint sensitivity generation

Instead of forward sensitivity generation which has been studied in the previous sections, it is also possible to use an adjoint approach. First order adjoint IND schemes are for example presented in [1, 2]. These schemes make use of the reverse mode of AD to propagate the sensitivity information backward through the integration process. Since the sensitivities of $N_x$ states with respect to $N_x + N_u + N_p$ parameters are needed, reverse propagation of the derivatives could be much more efficient. In Equation (3.6), describing the IFT method, it will be for example more efficient to first compute $\frac{\partial \phi}{\partial k}\left(\frac{\partial F}{\partial k}\right)^{-1}$ and then the product with $\frac{\partial F}{\partial w}$. Adjoint sensitivity generation can also be supported by solving the adjoint model [21]. This approach is for example presented in [9, 10] and it is discussed specifically for ERK methods in [5]. The formerly mentioned IND schemes generally have a better performance, since they allow one to reuse the results from major computations such as matrix factorizations [2]. Eventually, it is important to note that these adjoint techniques should certainly not be used in the case of intermediate outputs. And since continuous output is an important aspect of this thesis (see Chapter 5), only forward methods will be studied further.

## 3.5 Conclusion

Except for the VDE method, all the presented ways to generate the sensitivities are plausible to be used within the code generation of IRK methods. These methods are therefore tested thoroughly in the numerical experiments of Chapter 6. The

IFT-R method seems to be especially promising since it is able to compute the sensitivities up to machine precision without extra Newton iterations or an extra LU factorization.

# Chapter 4

# Simulation of Differential-Algebraic Equations

The goal of this chapter is to show that the IRK methods using code generation can also efficiently handle index-1 DAE systems. That is an important aspect because it is common to have a model containing algebraic equations like this:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), z(t)) \\ 0 &= g(t, x(t), z(t)) \end{aligned} \tag{4.1}$$

In this semi-explicit formulation, $z$ contains the $N_z$ algebraic states which are defined by the $N_z$ algebraic equations $0 = g(t, x, z)$. The important assumption here is that the DAE system is of index 1 which means that the matrix $\partial g/\partial z$ is invertible [26]. Often a more general model formulation is desirable. This chapter will therefore assume the following implicit DAE formulation:

$$\begin{aligned} 0 &= f(t, \dot{x}(t), x(t), z(t)) \\ 0 &= g(t, x(t), z(t)) \end{aligned} \tag{4.2}$$

This means that a model containing $M\dot{x} = \tilde{f}(t, x, z)$ with a mass matrix $M$ can be handled, but the function $f$ can also depend on $\dot{x}$ in a nonlinear way. The only extra assumption which will be made here is that the matrix $\partial f/\partial \dot{x}$ is invertible. Section 4.1 will present the extension of the IRK methods to integrate such an implicit DAE system and the corresponding implementation aspects. Also the application of the methods from Chapter 3 to compute the sensitivities in the case of a DAE system will be discussed briefly in Section 4.2.

## 4.1 Differential-Algebraic Equations

The generalization of the IRK methods to index-1 DAE systems can be done in two ways. The easiest way is to use only the differential states $x$ as independent variables and to reduce the DAE to the implicit ODE $f(t, \dot{x}, x, z(x)) = 0$. For nonlinear algebraic equations, this is however not efficient since this requires internal iterations

[53]. The alternative way is to explicitly deal with the algebraic equations in (4.2) and this is preferable here.

### 4.1.1 Formulation of the IRK methods

Similar to Equation (2.15) for an ODE system, an IRK method can be applied to the DAE system in (4.2) resulting in:

$$0 = f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i), \tag{4.3a}$$

$$0 = g(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i), \;\; i = 1, \ldots, s, \tag{4.3b}$$

$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i, \tag{4.3c}$$

$$0 = g(t_{n+1}, x_{n+1}, z_{n+1}), \tag{4.3d}$$

with $Z_i$ the stage values of the algebraic states, satisfying the algebraic equations. The nonlinear system consisting of (4.3a) and (4.3b) can be solved in a similar way as described for an explicit ODE in Section 2.4. The expression for $x_{n+1}$ is still explicit and (4.3d) makes sure that the new values for the states are consistent. Note that the latter equations are irrelevant in the case of a stiffly accurate IRK method such as the Radau IIA methods [33]. For these IRK methods $a_{si} = b_i$ for $i = 1, \ldots, s$ because $c_s = 1$, meaning that the endpoint is included into the quadrature nodes. The result is that for stiffly accurate IRK methods:

$$\begin{aligned} X_s &= x_n + h\sum_{j=1}^{s} a_{sj}k_j = x_{n+1} \\ Z_s &= z_{n+1} \end{aligned} \tag{4.4}$$

This avoids the need to solve the extra nonlinear system $0 = g(t_{n+1}, x_{n+1}, z_{n+1})$ in this case.

### 4.1.2 Implementation of the IRK methods

As already mentioned, (4.3a) and (4.3b) form the nonlinear system that is solved by an IRK method in the case of a DAE. Similar to Section 2.4, this system can be written in the more standard form $G(x_n, K) = 0$. In this formulation, $G$ consists of the functions in (4.3a) and (4.3b) for $i = 1, \ldots, s$ and the $K$-variables are defined as $K = (k_1, \ldots, k_s, Z_1, \ldots, Z_s)$. This nonlinear system can then be handled in the same way as described in Chapter 2 for an ODE. For example, the Jacobian $\frac{\partial G}{\partial K}(x_n, K)$ is

an $s(N_x + N_z) \times s(N_x + N_z)$ matrix that can be evaluated as follows:

$$\begin{pmatrix} \frac{\partial f_1}{\partial \dot{x}} + ha_{11}\frac{\partial f_1}{\partial x} & \cdots & ha_{1s}\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial z} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ ha_{s1}\frac{\partial f_s}{\partial x} & \cdots & \frac{\partial f_s}{\partial \dot{x}} + ha_{ss}\frac{\partial f_s}{\partial x} & 0 & \cdots & \frac{\partial f_s}{\partial z} \\ ha_{11}\frac{\partial g_1}{\partial x} & \cdots & ha_{1s}\frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial z} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ ha_{s1}\frac{\partial g_s}{\partial x} & \cdots & ha_{ss}\frac{\partial g_s}{\partial x} & 0 & \cdots & \frac{\partial g_s}{\partial z} \end{pmatrix}, \tag{4.5}$$

$f_i = f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i)$ and $g_i = g(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij}k_j, Z_i)$. The nonlinear system $G(x_n, K) = 0$ can then be solved using Algorithm 2.4 and the LU factorization will require $\sim \frac{2}{3}s^3(N_x + N_z)^3$ flops.

The main difference with the implementation of an IRK method for an ODE system, is that the algebraic states $z_{n+1}$ are found by solving a nonlinear system $0 = g(t_{n+1}, x_{n+1}, z_{n+1})$. The value for $z_{n+1}$ can first be predicted based on $z_n$ and the values $Z_i$, such that only a few Newton iterations are needed to solve this separate nonlinear system. A way to achieve this prediction is for example by using a slightly different formulation for the IRK method from the one in (4.3). Let us assume here that the values $x_n$ and $z_n$ for the states are consistent, meaning that $0 = g(t_n, x_n, z_n)$ holds with a certain accuracy. It is then possible to apply an IRK method to a DAE system as follows:

$$\begin{aligned} 0 &= f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij}k_j^x, z_n + h\sum_{j=1}^{s} a_{ij}k_j^z), \\ 0 &= g(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij}k_j^x, z_n + h\sum_{j=1}^{s} a_{ij}k_j^z), \;\; i = 1, \ldots, s, \\ x_{n+1} &= x_n + h\sum_{i=1}^{s} b_i k_i^x, \\ 0 &= g(t_{n+1}, x_{n+1}, z_{n+1}), \end{aligned} \tag{4.6}$$

where $k^x = (k_1^x, \ldots, k_s^x)$ and $k^z = (k_1^z, \ldots, k_s^z)$. A prediction for $z_{n+1}$ is then simply found as $z_{n+1}^{[0]} = z_n + h\sum_{i=1}^{s} b_i k_i^z$ and this prediction is often sufficiently accurate to allow only one Newton iteration to make the states consistent. Another advantage of this approach is that the principle of continuous output can then be used for the algebraic states in the same way as described in Section 2.2.2 for the differential states. The main disadvantage is the loss of sparsity in the Jacobian presented in (4.5), which can be exploited otherwise. If this particular structure is not being exploited, the IRK formulation in (4.6) can be used just as well.

It is however not a problem to compute a prediction for $z_{n+1}$, when the IRK formulation in (4.3) is used. Let us first show that (2.29) corresponds to the value at time $t_n + ch$ of the interpolating polynomial through the points $(t_n, x_n)$, $(t_n + c_1 h, X_1)$, $\ldots$, $(t_n + c_s h, X_s)$:

$$p(t_n + ch) = P_0(c)x_n + \sum_{i=1}^{s} P_i(c)X_i, \tag{4.7}$$

with the Lagrange interpolating polynomials $P_i(t)$:

$$\begin{aligned} P_0(t) &= (-1)^s \prod_{j=1}^{s} \frac{t - c_j}{c_j}, \\ P_i(t) &= \frac{t}{c_i} \prod_{\substack{j=1 \\ j\neq i}}^{s} \frac{t - c_j}{c_i - c_j} = \frac{t}{c_i} l_i(t), \quad i = 1, \ldots, s, \end{aligned} \tag{4.8}$$

with $l_i(t)$ defined as in (2.18). Using $X_i = x_n + h\sum_{j=1}^{s} a_{ij}k_j$, $c_0 = 0$, (2.22) and (4.8) in (4.7) results in:

$$\begin{aligned} p(t_n + ch) &= (-1)^s \prod_{j=1}^{s} \frac{c - c_j}{c_j} x_n + \sum_{i=1}^{s} \frac{c}{c_i} l_i(c)(x_n + h\sum_{j=1}^{s} a_{ij}k_j), \\ &= (\sum_{i=0}^{s} \prod_{\substack{j=0 \\ j\neq i}}^{s} \frac{c - c_j}{c_i - c_j}) x_n + h\sum_{i=1}^{s} \frac{c}{c_i} l_i(c) \sum_{j=1}^{s} a_{ij}k_j, \\ &= x_n + h\sum_{j=1}^{s} k_j \sum_{i=1}^{s} \frac{c}{c_i} l_i(c) \int_0^{c_i} l_j(\tau)\,\mathrm{d}\tau, \\ &= x_n + h\sum_{j=1}^{s} k_j \int_0^{c} l_j(\mu)\,\mathrm{d}\mu, \end{aligned} \tag{4.9}$$

with $\mu = \frac{c}{c_i}\tau$ and $\mathrm{d}\mu = \frac{c}{c_i}\mathrm{d}\tau$. This shows that the interpolating polynomial through the points $(t_n, x_n)$, $(t_n + c_1h, X_1)$, …, $(t_n + c_sh, X_s)$ is indeed the same as the collocation polynomial in Section 2.2. Similarly, the interpolating polynomial through the points $(t_n, z_n)$, $(t_n + c_1h, Z_1)$, …, $(t_n + c_sh, Z_s)$ can be defined for the algebraic states with the stage values $Z_i$ from (4.3). Equation (4.7) can then be used to achieve a prediction for $z_{n+1}$ when $c = 1$ or to evaluate the continuous output for the algebraic states:

$$z(t_n + ch) \approx P_0(c)z_n + \sum_{i=1}^{s} P_i(c)Z_i, \tag{4.10}$$

with $P_i(t)$ defined as in (4.8). It is important to note that it is not evident to expect consistent values for the states $x_n$ and $z_n$ to be known. In the case of a discontinuous jump in the control inputs, the differential states will still vary continuously. For algebraic states, it is possible to exhibit a jump due to the discontinuous change in the control inputs. Extra Newton iterations would be needed to achieve consistency $0 = g(t_n, x_n, z_n)$ with a certain accuracy and this at the beginning of every shooting interval since the control inputs might have changed. Multiple integration steps for this shooting interval will keep the states consistent, but the extra iterations can still be inefficient and therefore undesirable. The alternative is to only assure the states

to be consistent at the stages, i.e. the point $(t_n, z_n)$ is unavailable. In that case, the IRK formulation from (4.3) must be used and the interpolating polynomial for the algebraic states will be one order less than the one for the differential states.

### 4.1.3 Exploiting the structure

The Jacobian $\frac{\partial G}{\partial K}(x_n, K)$ from (4.5) definitely has some structure which can be exploited if the formulation in (4.3) is used for the IRK method. Let us rewrite this Jacobian as:

$$\begin{pmatrix} f_1^k & f_1^z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ f_s^k & 0 & \cdots & f_s^z \\ g_1^k & g_1^z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_s^k & 0 & \cdots & g_s^z \end{pmatrix}, \tag{4.11}$$

with $k = (k_1, \ldots, k_s)$, $f_i^k = \frac{\partial f_i}{\partial k}$, $f_i^z = \frac{\partial f_i}{\partial Z_i}$, $g_i^k = \frac{\partial g_i}{\partial k}$ and $g_i^z = \frac{\partial g_i}{\partial Z_i}$ according to (4.5). Applying Algorithm 2.4 to solve the nonlinear system $G(x_n, K) = 0$, the following linear system needs to be solved in every Newton iteration:

$$\begin{pmatrix} f_1^k & f_1^z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ f_s^k & 0 & \cdots & f_s^z \\ g_1^k & g_1^z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_s^k & 0 & \cdots & g_s^z \end{pmatrix} \begin{pmatrix} \Delta k \\ \Delta Z_1 \\ \vdots \\ \Delta Z_s \end{pmatrix} = \begin{pmatrix} -f_1 \\ \vdots \\ -f_s \\ -g_1 \\ \vdots \\ -g_s \end{pmatrix} \tag{4.12}$$

The last $sN_z$ linear equations can also be written as:

$$\begin{aligned} g_i^k \Delta k + g_i^z \Delta Z_i &= -g_i, \quad i = 1, \ldots, s \\ &\Downarrow \\ \Delta Z_i &= -(g_i^z)^{-1}(g_i + g_i^k \Delta k) \end{aligned} \tag{4.13}$$

Note that the inverse matrix $(g_i^z)^{-1}$ must exist, assuming the DAE system is of index 1. Using these expressions for $\Delta Z_i$, the linear system from (4.12) can be reduced to:

$$\begin{pmatrix} f_1^k - f_1^z (g_1^z)^{-1} g_1^k \\ \vdots \\ f_s^k - f_s^z (g_s^z)^{-1} g_s^k \end{pmatrix} \left( \Delta k \right) = \begin{pmatrix} -f_1 + f_1^z (g_1^z)^{-1} g_1 \\ \vdots \\ -f_s + f_s^z (g_s^z)^{-1} g_s \end{pmatrix} \tag{4.14}$$

Using these formulas to solve the nonlinear system, the dominating computational costs are:

| | |
|---|---|
| $s$ matrix inversions $(g_i^z)^{-1}$ | $O(sN_z^3)$ |
| $s$ matrix multiplications $f_i^z (g_i^z)^{-1}$ | $O(sN_x N_z^2)$ |
| $s$ matrix multiplications $(f_i^z (g_i^z)^{-1}) g_i^k$ | $O(s^2 N_x^2 N_z)$ |
| LU factorization of the matrix in (4.14) | $O(s^3 N_x^3)$ |

When the IRK method has a sufficient amount of stages $s$, there can indeed be a gain with respect to the computational complexity $O(s^3(N_x+N_z)^3)$ when no structure is exploited in the Jacobian of (4.11). Note that in the latter case the computational complexity is only $O(s^2(N_x + N_z)^2)$ for each Newton iteration in Algorithm 2.4 after the first one. Equation (4.14) shows that evaluating the right-hand side in each iteration requires more work when the structure in the Jacobian is exploited. Since the Jacobian in (4.12) is frozen after the first Newton iteration, the $s$ matrix inversions $(g_i^z)^{-1}$ and the matrix multiplications $f_i^z(g_i^z)^{-1}$ need to be computed only once. The dominating costs per Newton iteration are $O(s^2N_x^2)$ for Algorithm 2.5 to solve the linear system and $O(sN_xN_z)$ for the $s$ multiplications $(f_i^z(g_i^z)^{-1})g_i$ in the right-hand side of (4.14). Assuming $L$ Newton iterations are performed, the following comparison can be made between the dominating computational costs of the two approaches:

| IRK formulation (4.6) | | Exploiting the structure in (4.3) | |
|---|---|---|---|
| **First Newton iteration** | | | |
| LU factorization | $O(s^3N_x^3)$ | LU factorization | $O(s^3N_x^3)$ |
| | $O(s^3N_x^2N_z)$ | matrix multiplications | $O(s^2N_x^2N_z)$ |
| | $O(s^3N_xN_z^2)$ | | $O(sN_xN_z^2)$ |
| | $O(s^3N_z^3)$ | matrix inversions | $O(sN_z^3)$ |
| **Next Newton iterations** | | | |
| linear system | $O(s^2LN_x^2)$ | linear system | $O(s^2LN_x^2)$ |
| | $O(s^2LN_xN_z)$ | matrix-vector products | $O(sLN_xN_z)$ |
| | $O(s^2LN_z^2)$ | | |

This means that in the case of a relatively larger amount of stages $s$ and many algebraic states $N_z$, the structure in the Jacobian should be exploited as described here. Otherwise, either the formulation in (4.3) or the one in (4.6) can be used. Note that these statements are based on a purely asymptotic analysis.

## 4.2 Derivative Generation

Next to the new values of the states, the integration method also needs to deliver their sensitivities to the optimizer that handles the OCP. Chapter 3 already presented various ways to generate these sensitivities in the case of an ODE system and also evaluated the combination of these methods with an IRK method. It is rather straightforward to extend these principles to DAE systems. Let us again assume that $w_n = (x_n, u_n, p)$ denotes all the independent variables with respect to which the first derivatives of the results are needed. Different approaches to compute the matrix $\partial(x_{n+1}, z_{n+1})/\partial w_n$ will be described briefly.

### 4.2.1 Variational Differential Algebraic Equations (VDAE)

Section 3.3 presented the VDE as a way to compute the sensitivities by augmenting the ODE system. In the case of a DAE system, a similar approach can make use of the Variational Differential Algebraic Equations (VDAE). The VDAE can be obtained by differentiating the model equations in (4.2) with respect to $w_n$:

$$\begin{aligned} 0 &= \frac{\partial f}{\partial \dot{x}} \frac{\partial S_w^x}{\partial t} + \frac{\partial f}{\partial x} S_w^x + \frac{\partial f}{\partial z} S_w^z + \frac{\partial f}{\partial w_n}, \\ 0 &= \frac{\partial g}{\partial x} S_w^x + \frac{\partial g}{\partial z} S_w^z + \frac{\partial g}{\partial w_n}, \end{aligned} \tag{4.15}$$

with the sensitivity matrices $S_w^x = \partial x / \partial w_n$ and $S_w^z = \partial z / \partial w_n$. The solution of these equations is the needed sensitivity matrix $\partial(x, z)/\partial w_n$. However, this approach is not suitable for an IRK method for the same reasons as those stated in Section 3.3.

### 4.2.2 Internal Numerical Differentiation (IND)

Based on the discussion in Section 3.2, the sensitivities can also be computed using the principle of IND. One way to implement this is by using FD, which is not a very accurate way to generate the sensitivities. As mentioned before, the alternative is using AD and is called the IND-AD method in this text. The iterative scheme can be obtained by directly differentiating the equation describing a Newton iteration to solve the system $G(w_n, K) = 0$, combined with the update formulas (4.3c) and (4.3d). This results in

$$\begin{aligned} \frac{dK^{[i]}}{dw_n} &= \frac{dK^{[i-1]}}{dw_n} - M^{-1} \left( \frac{\partial G}{\partial w_n} + \frac{\partial G}{\partial K} \frac{dK^{[i-1]}}{dw_n} \right), \quad i = 1, \ldots, s, \\ \frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial x_n}{\partial w_n} + h \sum_{i=1}^{s} b_i \frac{dk_i^{[L]}}{dw_n}, \\ \frac{\partial z_{n+1}}{\partial w_n} &= -\frac{\partial g}{\partial z}^{-1} \left( \frac{\partial g}{\partial w_n} + \frac{\partial g}{\partial x} \frac{\partial x_{n+1}}{\partial w_n} \right), \end{aligned} \tag{4.16}$$

where the function $G(w_n, K)$ and the $K$-variables are defined as before and the matrix $M = \partial G / \partial K(w_n, K^{[0]})$. This IND-AD method is however costly when many directional derivatives need to be computed.

### 4.2.3 Implicit Function Theorem (IFT)

Finally, the method which seemed the most efficient in Chapter 3 in computing the sensitivities will also be preferable in the case of a DAE system and is denoted by

the IFT-R method. The approach is based on applying the IFT to (4.3), resulting in

$$\begin{aligned}\frac{dK}{dw_n} &= -\frac{\partial G}{\partial K}^{-1}\frac{\partial G}{\partial w_n},\\ \frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial x_n}{\partial w_n} + h\sum_{i=1}^{s} b_i \frac{dk_i}{dw_n},\\ \frac{\partial z_{n+1}}{\partial w_n} &= -\frac{\partial g}{\partial z}^{-1}\left(\frac{\partial g}{\partial w_n} + \frac{\partial g}{\partial x}\frac{\partial x_{n+1}}{\partial w_n}\right),\end{aligned} \tag{4.17}$$

with $G$ and $K$ defined as before. This IFT method can simply be combined with an IRK method to form an integrator with sensitivity generation. The idea of the IFT-R method is however to combine them in a way that the factorization of the matrices $\partial G/\partial K$ and $\partial g/\partial z$ only needs to be computed once per integration step. Algorithm 4.1 fully describes the implementation of one step of the IFT-R method for a DAE system such as (4.2).

**Algorithm 4.1** One step of the IFT-R method

**Input:** consistent $(x, z, u)_n$, $p$, initial $K^{[0]}$, LU factorization of matrices $M$ and $N$
**Output:** $(x, z)_{n+1}$ and $\frac{\partial (x,z)_{n+1}}{\partial w_n}$

$w_n \leftarrow (x_n, u_n, p)$
**if** $n = 0$ **then**
  $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[0]})$
  factorize $M$
  $K^{[0]} \leftarrow K^{[0]} - M^{-1}G(w_n, K^{[0]})$
**end if**
**for** $i = 1 \rightarrow L$ **do**
  $K^{[i]} \leftarrow K^{[i-1]} - M^{-1}G(w_n, K^{[i-1]})$
**end for**
$x_{n+1} \leftarrow x_n + h\sum_{i=1}^{s} b_i k_i$
predict $z_{n+1}$ (Section 4.1.2)
**if** $n = 0$ **then**
  $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$
  factorize $N$
**end if**
$z_{n+1} \leftarrow z_{n+1} - N^{-1}g(t_{n+1}, x_{n+1}, z_{n+1})$
$M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[L]})$ and $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$
factorize $M$ and $N$
compute sensitivities using $M$ and $N$ in (4.17)
initialize $K^{[0]}$ for next integration step (Section 2.5.2)
$n \leftarrow n + 1$

The Newton iterations of the IRK method reuse the matrix evaluation and LU factorization from the IFT in the previous integration step. The underlying assumption is that this Jacobian still serves as a good approximation. So instead

of evaluating and factorizing it at the first Newton iteration as in Algorithm 2.4, the factorized Jacobian from the previous IRK step can be used which needed to be computed for the sensitivities. The IFT-R method allows one to compute the sensitivities up to machine precision without the need for an extra LU factorization, except for the first integration step (when $n = 0$ in Algorithm 4.1). It is however useful to clarify what is meant here with the *first* integration step. In the case of single shooting, this is the first integration step in the first shooting interval. In the case of multiple shooting, this is the first integration step in every shooting interval.

## 4.3 Conclusion

This chapter first presented ways to apply the IRK methods to a DAE system and discussed the necessary modifications in the implementation. It showed that the principle of continuous output can also be used easily for the algebraic states. In the case of a DAE system, the Jacobian exhibits a clear sparsity structure which can be exploited. A more detailed look at the dominating computational costs however showed that not exploiting this structure can sometimes be nearly as efficient. The extension of the methods to generate the sensitivities for a DAE system is shown to be rather straightforward.

# Chapter 5

# Continuous Output

Some promising possibilities of auto generated IRK methods with continuous output have already been introduced in Chapter 1. This chapter will continue this discussion on the possible applications in Section 5.1 and will highlight some implementation aspects in Section 5.2. Section 5.3 briefly mentions the continuous extension of other RK methods than the collocation methods.

## 5.1 Motivation

As already mentioned in the introduction, the main motivation for implementing code generation of collocation methods instead of other IRK methods is the use of the continuous output for Moving Horizon Estimation (MHE). When using MHE to estimate the states and parameters for real-time control of a system, often different types of measurements are used. In the case of a mechatronic system with fast dynamics, often some variables need to be measured at a very high frequency while other measurements can only be performed at a rather low frequency. To be able to fully use these high frequency data without losing information, the integrator needs to provide the corresponding simulated data. The step size of the integrator then needs to be smaller than or equal to the sampling time of the measurements with the highest frequency. The availability of a continuous approximation of the states makes it possible to let the integrator take larger steps while still providing these data with a sufficient accuracy. For MHE, this can sometimes reduce the computation time spent in integration and sensitivity generation significantly.

The application of the continuous output, however, does not need to be restricted to MHE. For optimal control in general, it can e.g. happen that a constraint gets violated within a certain shooting interval while it is satisfied at the discretization points. If this constraint is of high importance, this can cause problems for the real system. The situation is depicted in Figure 5.1. The continuous output of the presented IRK methods makes it possible to check and respect such constraints on a finer grid with a relatively low cost in extra computation time.

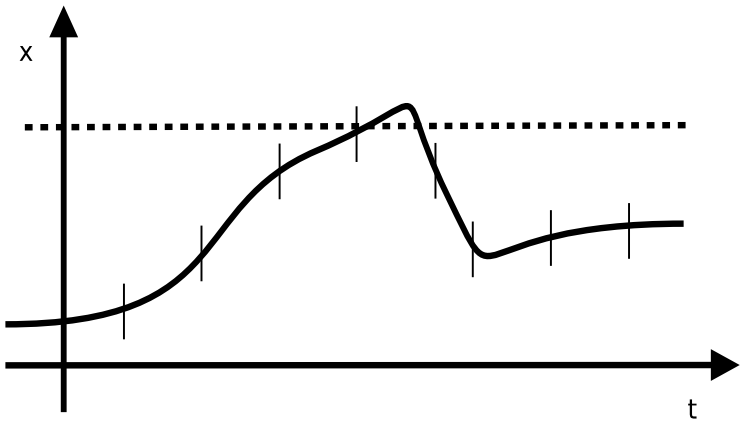



Figure 5.1: The red dotted line represents a certain constraint which gets violated in between two discretization points, denoted by the vertical bars.

A third motivation is the computation of least squares integrals $\int_0^T \|F(t,x,u)\|_2^2 \,\mathrm{d}t$ in the objective for NMPC or MHE (see Section 1.2). Using continuous output, the interval $[0,T]$ can be divided in arbitrarily small intervals which can lead to a much better approximation of the integral. But also other applications can still arise when this feature is made available in the auto generated integrators. This motivates us to make the implementation as general as possible.

## 5.2 Implementation

The model of the system can be augmented with a certain output function, representing the information that is needed on a finer grid than the one of the integrator. This output function depends on specific states. Let us assume the most general model which is discussed in this text, namely the implicit DAE of (4.2). The complete model formulation is then the following:

$$\begin{aligned} 0 &= f(t,\dot{x}(t),x(t),z(t)),\\ 0 &= g(t,x(t),z(t)),\\ y &= \psi(t,\dot{x}(t),x(t),z(t)), \end{aligned} \tag{5.1}$$

with $y$ a vector of $N_y$ outputs and $\psi$ the output function. The assumption here will be that the $N_y$ outputs are divided in $r$ groups:

$$\begin{aligned} y_1 &= \psi_1(t,\dot{x}(t),x(t),z(t))\\ &\dots\\ y_r &= \psi_r(t,\dot{x}(t),x(t),z(t)) \end{aligned} \tag{5.2}$$

Each group of outputs $y_i$ corresponds to a different output function $\psi_i$ and a grid discretizing one integration interval. The output function $\psi_i$ needs to be evaluated over this grid, using the continuous output. This formulation allows an efficient

implementation, while still being sufficiently general. Algorithm 5.1 describes how to evaluate the output functions in an efficient way. The grid discretizing $[t_n, t_{n+1}]$, that corresponds with function $\psi_i$ is defined by the points $0 \leq \tau_{i,1} < \tau_{i,2} < \ldots < \tau_{i,n_i} = 1$ for $i = 1, \ldots, r$. Meaning that $n_i$ is the number of points in the $i^{th}$ grid and that the endpoint is assumed to be $t_{n+1}$, i.e. the endpoint of the integration interval. Also note that $y_{i,j}$ represents the values of the $i^{th}$ group of outputs on the $j^{th}$ grid point. The IRK formulation in (4.3) is used with the continuous output of the differential and algebraic states defined by respectively (2.29) and (4.10).

**Algorithm 5.1** Evaluation of the output functions

**Input:** consistent $(x, z)_n$ and $(x, z)_{n+1}$, $K = (k_1, \ldots, k_s, Z_1, \ldots, Z_s)$
**Output:** $y_{i,j}$ for $i = 1, \ldots, r$ and $j = 1, \ldots, n_i$

$\dot{x}_{n+1} \leftarrow \sum_{i=1}^{s} k_i l_i(1)$
**for** $i = 1 \rightarrow r$ **do**
  **for** $j = 1 \rightarrow n_i - 1$ **do**
    $\dot{x}_{\text{temp}} \leftarrow \sum_{m=1}^{s} k_m l_m(\tau_{i,j})$
    $x_{\text{temp}} \leftarrow x_n + h \sum_{m=1}^{s} k_m \int_0^{\tau_{i,j}} l_m(\tau)\, \mathrm{d}\tau$ with $l_m(t)$ defined in (2.18)
    $z_{\text{temp}} \leftarrow P_0(\tau_{i,j}) z_n + \sum_{m=1}^{s} P_m(\tau_{i,j}) Z_m$ with $P_m(t)$ defined in (4.8)
    $y_{i,j} \leftarrow \psi_i(t_n + \tau_{i,j} h, \dot{x}_{\text{temp}}, x_{\text{temp}}, z_{\text{temp}})$
  **end for**
  $y_{i,n_i} \leftarrow \psi_i(t_{n+1}, \dot{x}_{n+1}, x_{n+1}, z_{n+1})$
**end for**

# only states on which $\psi_i(t, \dot{x}(t), x(t), z(t))$ depends are calculated

If the outputs are used in the optimization problem, then their sensitivities are also needed. Algorithm 5.2 therefore describes how to generate the sensitivities $\partial y / \partial w_n$ of the continuous output with $w_n = (x_n, u_n, p)$.

**Algorithm 5.2** Sensitivity generation for the outputs

**Input:** $\frac{dK}{dw_n}$, $\frac{\partial (x,z)_{n+1}}{\partial w_n}$
**Output:** $\frac{\partial y_{i,j}}{\partial w_n}$ for $i = 1, \ldots, r$ and $j = 1, \ldots, n_i$

$\frac{\partial \dot{x}_{n+1}}{\partial w_n} \leftarrow \sum_{i=1}^{s} \frac{dk_i}{dw_n} l_i(1)$
**for** $i = 1 \rightarrow r$ **do**
  **for** $j = 1 \rightarrow n_i - 1$ **do**
    $\frac{\partial \dot{x}_{\text{temp}}}{\partial w_n} \leftarrow \sum_{m=1}^{s} \frac{dk_m}{dw_n} l_m(\tau_{i,j})$
    $\frac{\partial x_{\text{temp}}}{\partial w_n} \leftarrow \frac{\partial x_n}{\partial w_n} + h \sum_{m=1}^{s} \frac{dk_m}{dw_n} \int_0^{\tau_{i,j}} l_m(\tau)\, \mathrm{d}\tau$
    $\frac{\partial z_{\text{temp}}}{\partial w_n} \leftarrow \sum_{m=1}^{s} P_m(\tau_{i,j}) \frac{dZ_m}{dw_n}$
    $\frac{\partial y_{i,j}}{\partial w_n} \leftarrow \frac{\partial \psi_i}{\partial \dot{x}} \frac{\partial \dot{x}_{\text{temp}}}{\partial w_n} + \frac{\partial \psi_i}{\partial x} \frac{\partial x_{\text{temp}}}{\partial w_n} + \frac{\partial \psi_i}{\partial z} \frac{\partial z_{\text{temp}}}{\partial w_n}$
  **end for**
  $\frac{\partial y_{i,n_i}}{\partial w_n} \leftarrow \frac{\partial \psi_i}{\partial \dot{x}} \frac{\partial \dot{x}_{n+1}}{\partial w_n} + \frac{\partial \psi_i}{\partial x} \frac{\partial x_{n+1}}{\partial w_n} + \frac{\partial \psi_i}{\partial z} \frac{\partial z_{n+1}}{\partial w_n}$
**end for**

# only sensitivities of states on which $\psi_i$ depends are calculated

Remember that Section 3.4 briefly mentioned the possibility of adjoint sensitivity generation. In this context of the use of continuous output, forward propagation of the derivatives is however much more efficient than reverse propagation.

## 5.3 Continuous extension of methods

The large family of RK methods can be divided in a continuous and a discrete group. The continuous group consists of the collocation methods, which are mainly studied here. While the discrete group consists of all the other RK methods. It is however possible to construct an interpolant for these RK methods. It is then preferable to minimize the needed number of extra function evaluations to obtain an interpolant of a specific order. The values of $k_i = f(t_n + c_i h, X_i)$ for $i = 1, \ldots, s$ and of $x_n$ and $x_{n+1}$ are already available to be used. A very general procedure to construct interpolants for explicit, implicit or semi-implicit RK methods is described in [20]. The approach is also illustrated for three explicit RK methods. It seems often possible to obtain a continuous extension of reasonable order for these explicit methods without extra function evaluations. An interpolation formula can be derived similarly for a semi-implicit method, as is illustrated in [41]. It shows that the $3^{\text{rd}}$ order ESDIRK method from Section 2.1.3 can also provide continuous output of order 3 using the values of $x_n$, $k_1$, $k_3$ and $k_4$.

## 5.4 Conclusion

In addition to MHE with multi-rate measurements, also other possible applications motivated a quite general formulation for the continuous output feature of the presented integration methods. The chapter however also indicated that this formulation still allows an efficient implementation of the generation of these extra outputs and their sensitivities.

# Chapter 6

# Numerical Experiments

The purpose of this chapter is to study how all the theoretical methods and ideas, described in the previous chapters, eventually pay off in numerical efficiency. The subject is the implementation of code generation for Implicit Runge-Kutta methods with Continuous output (IRKC) with efficient sensitivity generation. The implementation still allows to change some parameters such as the order of the method, the Butcher table or the number of Newton iterations, but this chapter will include guidelines to set these parameters. Everything is newly implemented, including the export of linear solvers, and the whole software is integrated in the ACADO code generation tool (see Appendix C).

All the numerical experiments presented in this chapter are run on an ordinary computer (Intel P8600 3MB cache, 2.40 GHz, 64-bit Ubuntu 10.04 LTS with Clang 3.1) and the time measurements are done using the `gettimeofday` function.

The chapter starts by describing the used test problems for ODE systems (Section 6.1) and for DAE systems (Section 6.2), without worrying about physical units for our purely numerical purposes. The influence of some standard parameters will be studied in Section 6.3. Section 6.4 then compares the different methods from Chapter 3 to generate the sensitivities and will indicate the most efficient method. Some improvements in the initialization of the variables of the collocation method will be tested in Section 6.5, based on the discussion in Section 2.5. Section 6.6 discusses numerical experiments, regarding the extension of the methods to DAE systems. Eventually, the performance of the auto generated IRK methods will be illustrated in Section 6.7 with attention for the continuous output and Section 6.8 will briefly illustrate the usage within real-time optimal control.

## 6.1 Test Problems with ODE Systems

Some different test systems will be used to be able to make well-founded conclusions from the numerical experiments. In this section, the test problems described by an ODE system will be introduced briefly. They mainly differ in their number of equations (which is the number of differential states) and in the complexity of these

equations. The ODE systems will be presented, starting with the smallest system until the largest system.

### 6.1.1 Van der Pol oscillator

The Van der Pol oscillator is described by a stiff ODE system and it is frequently used as a test problem (e.g. in [33]). The Van der Pol equation is one of the simplest nonlinear equations describing an electrical circuit and it can be written as an ODE system

$$\begin{cases} \dot{y_1} = y_2 \\ \dot{y_2} = ((1 - y_1^2)y_2 - y_1)/\varepsilon \end{cases} \\ y_1(0) = 2, y_2(0) = 0, \varepsilon = 0.01 \tag{6.1}$$

This system can easily be integrated for large values of $\varepsilon$. For smaller values of $\varepsilon$, the problem can however become very stiff. In this chapter, the Van der Pol oscillator will always be used with $\varepsilon = 0.01$. At the initial values $(y_1(0) = 2, y_2(0) = 0)$, the Jacobian of the system has 2 real eigenvalues equal to $-0.3337$ and $-299.6663$. The stiffness ratio of the system is therefore initially equal to $299.6663/0.3337 = 898$ (see Section 2.3.1), which means that the system is relatively stiff.

### 6.1.2 An overhead crane ODE model

This test problem consists of an ODE system modeling an overhead crane (trolley and cable) similar to [58]. The version of the model that will be used here as a test problem, has the following 8 differential states:

$$\begin{aligned} x_T &: \text{position of the trolley} \\ v_T &: \text{velocity of the trolley} \\ x_L &: \text{length of the cable} \\ v_L &: \text{velocity of the cable} \\ \phi &: \text{the excitation angle} \\ \omega &: \text{angular velocity} \\ u_T &: \text{input to the trolley velocity controller} \\ u_L &: \text{input to the cable velocity controller} \end{aligned} \tag{6.2}$$

and two control inputs:

$$\begin{aligned} u_T^{\text{dot}} &: \text{change of the input to the trolley velocity controller} \\ u_L^{\text{dot}} &: \text{change of the input to the cable velocity controller} \end{aligned} \tag{6.3}$$

The ODE system is then the following

$$
\begin{cases}
\dot{x_T} = v_T \\
\dot{v_T} = -\frac{1}{\tau_1} v_T + \frac{a_1}{\tau_1} u_T \\
\dot{x_L} = v_L \\
\dot{v_L} = -\frac{1}{\tau_2} v_L + \frac{a_2}{\tau_2} u_L \\
\dot{\phi} = \omega \\
\dot{\omega} = \frac{1}{x_L}(-g\sin(\phi) - a_T\cos(\phi) - 2v_L\omega - c\frac{\omega}{m x_L}) \\
\dot{u_T} = u_T^{\text{dot}} \\
\dot{u_L} = u_L^{\text{dot}}
\end{cases} \tag{6.4}
$$

$$
x_T(0) = 0.5, v_T(0) = 0.1, x_L(0) = 0.7, v_L(0) = -0.1,
$$
$$
\phi(0) = 0.5, \omega(0) = -0.07, u_T(0) = 0.2, u_L(0) = -0.3
$$

In these equations, the parameters $\tau_1 \approx 0.0128$ and $\tau_2 \approx 0.0247$ are time constants and the parameters $a_1 \approx 0.0474$ and $a_2 \approx 0.0341$ both denote a gain. The parameter $g = 9.81$ denotes the gravitational constant, $m = 1318$ is the mass of the pendulum and $c = 0$ is the damping constant for the motion of the pendulum (so there is no damping). Note that the stiffness ratio of the system is here initially equal to 547 and can increase to $\sim 5000$, so this test system is also quite stiff. Figure 6.1 presents the eigenvalues of the Jacobian evaluated at the initial values.

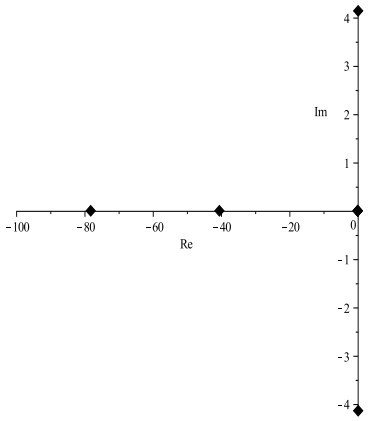



Figure 6.1: The eigenvalues of the Jacobian of the crane model for its initial values.

### 6.1.3 A kite ODE model

This test problem consists of one of the models developed for MPC of tethered planes for wind power generation [59], in the context of the ERC Highwind Project [51]. It consists of a pointmass model for the plane kinematics and the tether is modeled as an elastic rod. The use of a pointmass model means that the translation and the roll are considered, but the rotations do not need to be described. The assumption made is that the plane is always aligned with the relative wind. The control inputs in this pointmass model are the rate of change of the roll $\Psi^{\text{dot}}$ and of the aerodynamic lift coefficient $C_L^{\text{dot}}$. The benefit of such a model instead of a rota-

tional model is that the equations are less complex, but the model is also less realistic.

As already mentioned, this model for the plane kinematics needs to be extended with a tether model. In this test system, the tether is modeled as an elastic rod which results in an ODE formulation. If the tether would be modeled as an infinitely rigid rod, resulting in a simple constraint on the position, then an index-3 DAE would be obtained (see Section 6.2.2). The elastic version of the model is also closer to reality than introducing such a constraint, but the obtained ODE system is stiff. At the initial values, the stiffness ratio of the system is equal to 45 and the eigenvalues are located as in Figure 6.2.

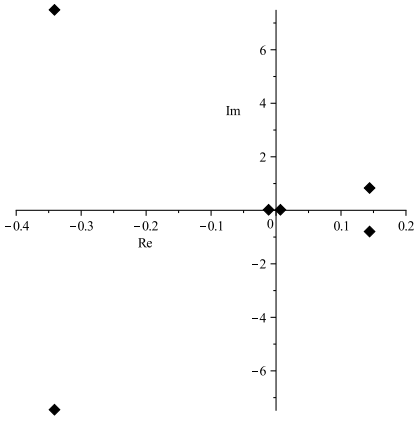



Figure 6.2: The eigenvalues of the Jacobian of the kite ODE model for its initial values.

Eventually a third control input is added, which is the second derivative of the rod length $r$ with respect to time. The test problem has 11 differential states:

$$\begin{aligned}(x, y, z) &: \text{position of the kite}\\ (v_x, v_y, v_z) &: \text{velocity of the kite}\\ r &: \text{rod length}\\ r^{\text{dot}} &: \text{rate of change of } r\\ n &: \text{winding number}\\ \Psi &: \text{the roll angle}\\ C_L &: \text{the aerodynamic lift coefficient}\end{aligned} \tag{6.5}$$

## 6.2 Test Problems with DAE Systems

Also two different test DAE systems will be used and they are described in this section. They again differ in their size, i.e. the number of equations but also in the number of algebraic states.

### 6.2.1 An inverted pendulum DAE model

This DAE system models a planar pendulum, such as depicted in Figure 6.3. The model consists of the Newton-Euler-equations and some kinematical constraints [56]. It is naturally formulated as a DAE of index 3, but using index reduction it can be transformed into an index-1 DAE system [8]. The eventual model has 6 differential states

$$\begin{aligned} (x, y, \alpha) &: \text{position of the pendulum,} \\ (v_x, v_y, v_\alpha) &: \text{velocity of the pendulum,} \end{aligned} \tag{6.6}$$

5 algebraic states

$$\begin{aligned} (a_x, a_y, a_\alpha) &: \text{acceleration of the pendulum,} \\ (F_x, F_y) &: \text{resulting force,} \end{aligned} \tag{6.7}$$

and one control input $u$ which denotes a force in the x-direction.

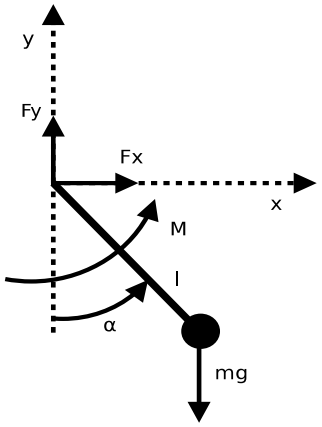



Figure 6.3: A simple planar pendulum.

The DAE system can then be written in the following semi-explicit form:

$$
\begin{cases}
\dot{x} = v_x \\
\dot{y} = v_y \\
\dot{\alpha} = v_\alpha \\
\dot{v}_x = a_x \\
\dot{v}_y = a_y \\
\dot{v}_\alpha = a_\alpha \\
0 = ma_x - (F_x + u) \\
0 = ma_y + mg - F_y \\
0 = Ia_\alpha - M - (F_x + u)y + F_y x \\
0 = a_x + v_y v_\alpha + y a_\alpha \\
0 = a_y - v_x v_\alpha - x a_\alpha
\end{cases}
\tag{6.8}
$$

$$x(0) = 1, y(0) = -5, \alpha(0) = 1, v_x(0) = 0.1, v_y(0) = -0.5, v_\alpha(0) = 0.1,$$
$$a_x(0) = -1.5, a_y(0) = -0.3, a_\alpha(0) = -0.3, F_x(0) = -3, F_y(0) = 19$$

The parameters in this formulation are the mass $m = 2$, the applied torque $M = 3.5$, the moment of inertia $I = 0.1$ and the gravitational constant $g = 9.81$. It is a rather easy model to integrate, but still a useful test system since the nonlinearities are contained in the algebraic equations.

### 6.2.2 A kite DAE model

This test system is similar to the one in Section 6.1.3, but here the tether is modeled as an infinitely rigid rod which results in a constraint on the position. The DAE which is then obtained is of index 3 but can be reduced to an index-1 DAE system. In addition, the model is also a more sophisticated one with 18 differential states, 1 algebraic state and 4 control inputs. The differential states consist of:

$$
\begin{aligned}
(x, y, z) &: \text{position of the kite} \\
(v_x, v_y, v_z) &: \text{velocity of the kite} \\
(q_0, q_1, q_2, q_3) &: \text{attitude of the kite} \\
(w_1, w_2, w_3) &: \text{rotation speed} \\
r &: \text{kite distance} \\
v_r &: \text{kite velocity} \\
E &: \text{energy} \\
R &: \text{regularization} \\
n &: \text{winding number}
\end{aligned}
\tag{6.9}
$$

the control inputs are:

$$
\begin{aligned}
v_r^{\text{dot}} &: \text{kite acceleration} \\
u_1 &: \text{pitch control} \\
u_2 &: \text{roll control} \\
u_3 &: \text{yaw control}
\end{aligned}
\tag{6.10}
$$

and the algebraic state $\lambda$ is a Lagrange multiplier for the cable.

## 6.3 Choice of the Standard Parameters

The goal of this section is to have an idea about suitable values for the standard parameters such as the order of the IRKC method, the step size $h$ and the number of iterations $L$ in the Newton method from Algorithm 2.4.

### 6.3.1 Experiments

The test systems from Section 6.1 will be used, for which all the combinations of the following values for these standard parameters are tested:

| h | order | L |
|---|---|---|
| 0.1 | 2 | 1 |
| 0.05 | 4 | 2 |
| 0.02 | 6 | 3 |
| 0.01 | | |
| 0.005 | | |

The crane and kite models will be integrated over the time interval $[0, 1]$, the Van der Pol oscillator is integrated over the time interval $[0, 0.5]$.

### 6.3.2 Results and Discussion

The results for the developed integrators are validated using the results from the `ode15s` integrator of Matlab with a sufficiently small relative tolerance of $10^{-10}$. To obtain a measure for the efficiency, the idea here is to compare the following:

- the *mean relative error*: the maximum relative error over all the states, but the mean of these results over the different grid points
- the *total simulation time*: the computation time needed to integrate the system over the complete time interval

The numerical results will be presented in a work-precision diagram, with a logarithmic scale for both axes such as in [33]. The horizontal axis will show the *mean relative error* in a reversed direction. This means that the more a point is to the right, the lower is the *mean relative error* or the higher is the precision. And the vertical axis will show the *total simulation time*. The obtained precision is essentially determined by the value for the step size $h$. So it is impossible to choose a suitable value for this step size without knowing the required accuracy and the problem on which the method is used. The results for different values of the step size $h$ will be presented as one polygonal line in the work-precision diagram. This allows a fair comparison between the different methods, since a continuous approximation of the solution is obtained for any value of $h$ anyway.

Different work-precision diagrams are presented: 3 diagrams for the order of the method and 3 diagrams for the number of Newton iterations. The horizontal axis will always have a range from 1 to $10^{-6}$. The assumption here is that a relative error smaller than $10^{-6}$ is excessive and an error greater than 1 means that the results are worthless. The work-precision diagrams for the crane can for example be found in Figure 6.4. It seems that the order of 6 for the IRKC method is already too high. The required amount of work is namely higher for this order 6 method than for the order 2 and 4 methods to achieve a similar precision. From the 3 lower diagrams, it is clear that the number of Newton iterations is more important for a higher order method. The order 2 method for example appears to achieve the same precision when more than 1 Newton iteration is performed. This means that performing more Newton iterations will only increase the simulation time in such a case. For the order 4 and 6 Gauss methods, it seems possible to achieve a better precision by performing more than one Newton iteration.

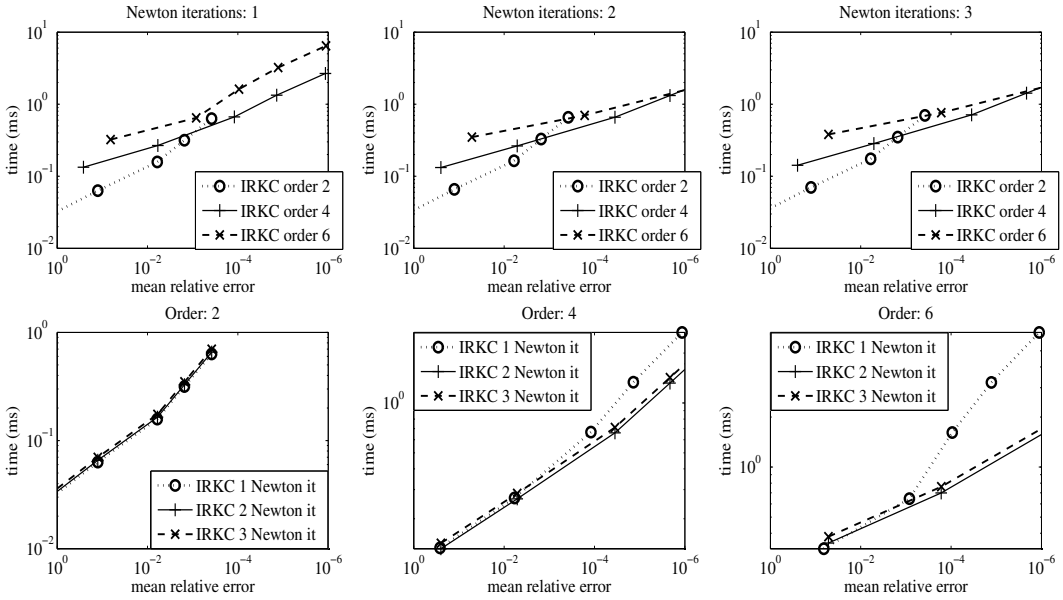



Figure 6.4: Crane model: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method.

An important observation is also that the slope of a curve should express the order of the corresponding formula in the sense that lower order methods are steeper than higher order methods [33]. This is confirmed in Figure 6.4. Every order 2, 4 or 6 for the IRKC method has an according range of precision in which it is the most efficient method. The order 2 IRKC method namely seems the most efficient when a small precision is sufficient and the order 4 method is the most efficient method when a higher precision is needed. The order 6 method will eventually be the most efficient

one when looking at even higher precisions, which again means it is irrelevant for the targeted real-time applications. One Newton iteration is enough for the order 2 method but it is preferable to perform 2 Newton iterations for the order 4 and order 6 method because of the large gain in efficiency. As more Newton iterations do not deteriorate the performance of the order 2 methods, from here on always 2 Newton iterations will be used. All the previous conclusions are confirmed by the similar results for the Van der Pol oscillator or the kite model, which can be found in Section D.1 of Appendix D.

## 6.4 Efficient Derivative Generation

The goal of this section is to compare the efficiency of different methods to generate the needed sensitivities, described in Chapter 3.

### 6.4.1 Experiments

The test systems from Section 6.1 will again be used for these numerical experiments. From the previous section, it is quite clear that it is preferable to use 2 Newton iterations. It is also clear from these experiments that an order of 6 for the IRKC method is already too high. The order 2 and 4 Gauss methods will therefore be used here to obtain a clear overview. All the combinations of the following values for the parameters are tested here:

| h | order |
|---|---|
| 0.1 | 2 |
| 0.05 | 4 |
| 0.02 | |
| 0.01 | |
| 0.005 | |

The following methods to generate the sensitivities of the solution will be tested:

- FD: Internal Numerical Differentiation, using Finite Differences (Section 3.2.1)
- IND-AD: Internal Numerical Differentiation, using Algorithmic Differentiation (Section 3.2.2)
- IFT: making use of the Implicit Function Theorem (Section 3.1)
- IFT-R: an efficient alternative implementation of the IFT method (Section 3.1.1)

All these different methods result in different implementations of the IRKC method. Only the approach using the VDE will not be included in the comparison because it does not compete with the other approaches in combination with an IRK

method. The ERK4 integrator will be used as a reference for the efficiency of these implementations. This was the only integrator available in the code generation tool of ACADO before the work presented in this text.

### 6.4.2 Results and Discussion

In addition to work-precision diagrams for the state values, similar diagrams for the sensitivities will be presented. The problem when studying the precision of the sensitivities is however that the exact derivatives of the numerical results for the discretized system are unknown. A reasonable assumption which is made here is that the IFT method (Section 3.1) computes the sensitivities much more accurately than the other methods. The IFT method namely computes a new evaluation and factorization of the Jacobian to generate these sensitivities using an exact expression. This is in contrast to the FD and IND-AD method which both reuse the previously calculated factorization in some Newton iterations. Also remember that Section 3.2.2 has proven that the sensitivities generated by the IND-AD method eventually converge to those generated by the IFT method. The mean relative error of the sensitivities is therefore defined here with respect to those computed with the IFT method. For the states, the accuracy is still determined using the `ode15s` integrator of Matlab.

The numerical results for the kite ODE model are presented in Figure 6.5. Note that the execution time always consists of the complete integration with sensitivity generation. The upper diagrams for the states show that the IFT-R method results in the most efficient implementation. Differences between the methods for these diagrams are mainly caused by differences in the computational complexity. It is however clear that the IND-AD and the IFT-R method deliver much more accurate sensitivities than the FD method. Even for smaller values of the step size $h$, the precision of the sensitivities computed with the FD method is much smaller.

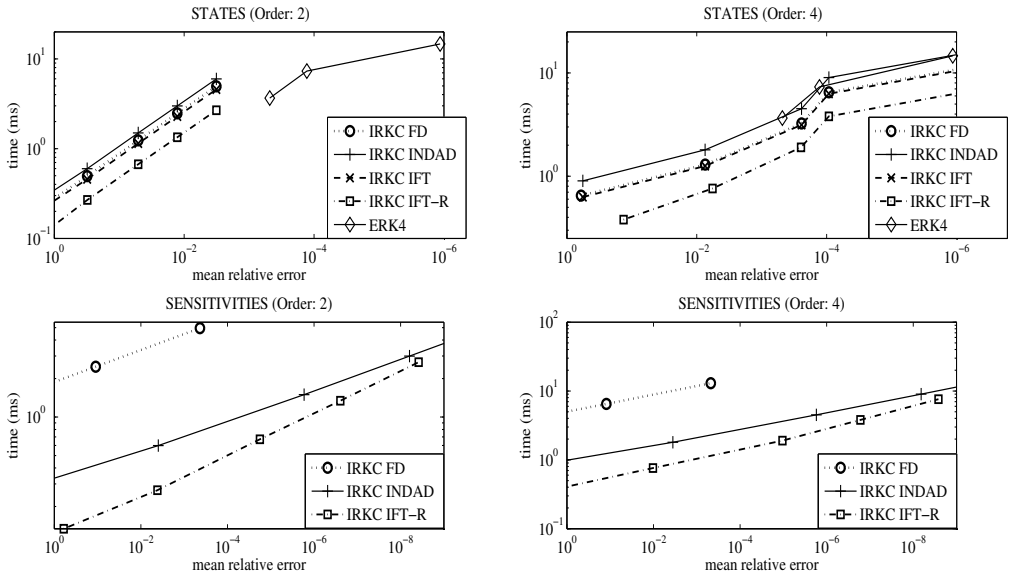



Figure 6.5: Elastic kite model: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities.

The most important conclusion from these experiments is to use the IFT-R implementation described in Section 3.1.1 to efficiently generate the needed sensitivities. It is also interesting to note that the IRKC with IFT-R method of order 4 is even more efficient than the ERK4 method for the kite example. The order 2 method is only more efficient when a small precision is sufficient. The ERK4 method is namely not even stable for a larger step size because of the stiffness of these test systems. The same conclusions are made for the Van der Pol oscillator or the crane model, since the results are similar (see Section D.2). Table 6.1 for example shows the average computation time per integration step for the crane model.

It can also be interesting to have a look at the time spent in the LU factorization, the solution of a linear system reusing this factorization, evaluations of the Jacobian and other computations and this for the different methods. Table 6.2 shows this for the $4^{\text{th}}$ order Gauss method. The set of other computations comprises of all the extra computations related to the IRK method or the sensitivity generation and can be considered as overhead. This confirms the efficiency of the IFT-R method with respect to the other implementations.

| order IRK | FD | IND-AD | IFT | IFT-R |
|---|---|---|---|---|
| 2 | 4.8 µs | 4.8 µs | 3.0 µs | 2.2 µs |
| 4 | 15.7 µs | 23.3 µs | 12.7 µs | 8.0 µs |
| 6 | 38.0 µs | 61.8 µs | 33.8 µs | 21.2 µs |

Table 6.1: Average computation time per integration step for overhead crane example.

| | FD | IND-AD | IFT | IFT-R | |
|---|---|---|---|---|---|
| LU factorization | 20.6 % | 16.1 % | 53.6 % | 44.2 % | 3.5 µs |
| linear system | 39.6 % | 27.8 % | 27.3 % | 41.9 % | 3.4 µs |
| evaluation Jacobian | 1.9 % | 3.7 % | 5.3 % | 5.1 % | 0.4 µs |
| other computations | 37.9 % | 52.4 % | 13.9 % | 8.8 % | 0.7 µs |
| total | 15.7 µs | 23.3 µs | 12.7 µs | 100.0 % | 8.0 µs |

Table 6.2: Composition of the average computation time per integration step for the $4^{\text{th}}$ order Gauss method on crane model.

## 6.5 Initialization of the Variables

This section will discuss some numerical experiments which will test the possibilities described in Section 2.5 to better initialize the variables of the collocation method. The distinction will be made between the initialization of the first step and of a subsequent step. The test systems from Section 6.1 will again be used.

### 6.5.1 Initialization of the First Step

Let us study the impact of doing one or more extra Newton iterations for the first integration step on the total simulation time and on the mean relative error of the results over the complete time interval (see Section 2.5.1).

#### Experiments

For these numerical experiments, only the IRKC with IFT-R method of orders 2 and 4 will be used because of the previous results. Different numbers of extra Newton iterations for the first integration step $L^{\text{init}}$ will be tried, to find out which value results in the best efficiency. All the combinations of the following values for the different parameters will be used:

| h | $L^{\text{init}}$ | order |
|---|---|---|
| 0.1 | 0 | 2 |
| 0.05 | 1 | 4 |
| 0.02 | 2 | |
| 0.01 | | |
| 0.005 | | |

The ERK4 integrator will again be used as a reference for the efficiency of the different implementations of the IRKC method.

### Results and Discussion

Figure 6.6 shows 2 work-precision diagrams for the results of the order 2 and order 4 method. It seems that the extra Newton iterations have no noticeable influence on the efficiency of the IRKC method. The achieved precision and the total simulation time is very similar for the 3 different cases. The same conclusions are made for the Van der Pol oscillator or the kite model. Just to be safe, one extra Newton iteration could however be performed during the first integration step.

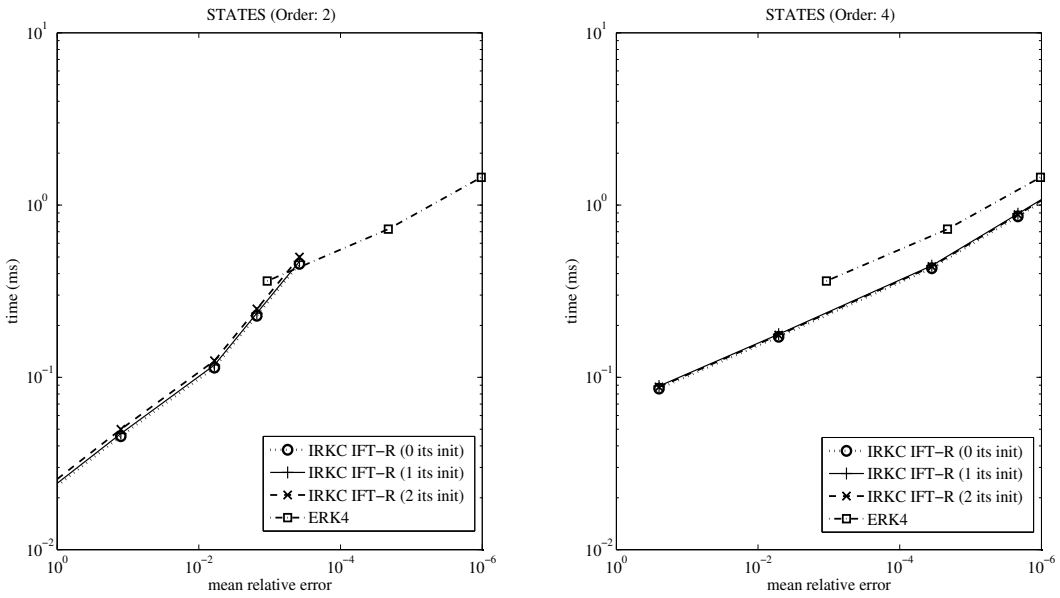



Figure 6.6: Crane model: 2 work-precision diagrams for the order 2 and order 4 IRKC with IFT-R method to study the impact of performing 0, 1 or 2 extra Newton iterations during the first integration step.

### 6.5.2 Initialization of Subsequent Steps

The goal of these experiments is to study the impact of using the continuous output to initialize the variables of the IRKC method for a next integration step (see Section 2.5.2).

#### Experiments

These numerical experiments will use only the IRKC with IFT-R method of order 4. Section 2.5.2 already showed that for the order 2 Gauss method, there is no difference between the two initialization methods which are described there. Note that in the previous experiments, the number of Newton iterations per integration step $L$ was equal to 2 because of the results in Section 6.3. However, this number of Newton iterations will again be altered to see if it could be more efficient to do only 1 Newton iteration in combination with a better initialization. The following values for the different parameters will therefore be used:

| h | L |
|---|---|
| 0.1 | 1 |
| 0.05 | 2 |
| 0.02 | |
| 0.01 | |
| 0.005 | |

Two variants of the IRKC with IFT-R method are tested with the following initialization methods:

- Warm-start: used in the previous experiments, i.e. the variables are initialized with the converged values from the previous integration step
- Extrapolation: the new implementation which uses the continuous output of the collocation method to initialize the variables for the next integration step

#### Results and Discussion

The results for the crane example can be found in Figure 6.7. In the case of 2 Newton iterations per integration step, the use of Extrapolation has almost no advantage. If only 1 Newton iteration per integration step is performed, then there is a big increase in the precision of the calculated states when using Extrapolation to initialize the variables. However, performing only 1 Newton iteration still appears to be less efficient than performing 2 Newton iterations per step.

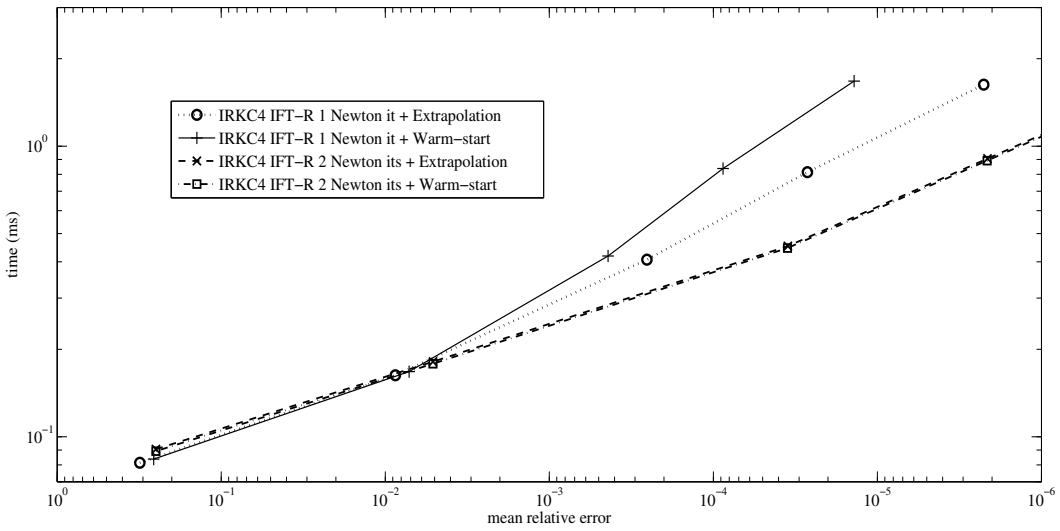



Figure 6.7: Crane model: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables.

The conclusion from these results is that Extrapolation should be used to initialize the variables in the next step. Because it is relatively cheap and it strongly improves the efficiency of the methods if only 1 Newton iteration per integration step is performed. The same conclusions are made from the results for the Van der Pol oscillator or the kite model (see Section D.3).

## 6.6 Solving the Algebraic equations

Section 4.1.2 discussed how to keep the differential and algebraic states consistent during the integration of a DAE system. In addition to the collocation equations, a separate nonlinear system consisting of the algebraic equations then needs to be solved. The section also presented how to compute an accurate prediction for the algebraic states to reduce the needed number of Newton iterations for this second nonlinear system. This section will study the impact of these Newton iterations to solve the algebraic equations. Note that exploiting the structure in the Jacobian as described in Section 4.1.3, is not really beneficial for the small test problems discussed here. In agreement with the results from Section 6.4, the IFT-R method will still be used to generate the sensitivities.

### 6.6.1 Experiments

The test systems from Section 6.2 will be used with the following values for the parameters:

| h | order | alg its |
|---|---|---|
| 1.0 | 2 | 0 |
| 0.5 | 4 | 1 |
| 0.2 | 6 | 2 |
| 0.1 | | |
| 0.05 | | |

The pendulum and kite DAE model are integrated over the time interval $[0, 10]$.

### 6.6.2 Results and Discussion

Figure 6.8 shows the results for the pendulum in one work-precision diagram. Similar to the results in Figure 6.4, it seems that also the number of Newton iterations for the algebraic equations is more important for a higher order method. The order 2 method even appears to achieve the same precision when the prediction for the algebraic states is used without any extra Newton iterations. For the order 4 and order 6 Gauss methods, one Newton iteration for the separate nonlinear system however strongly increases the precision.

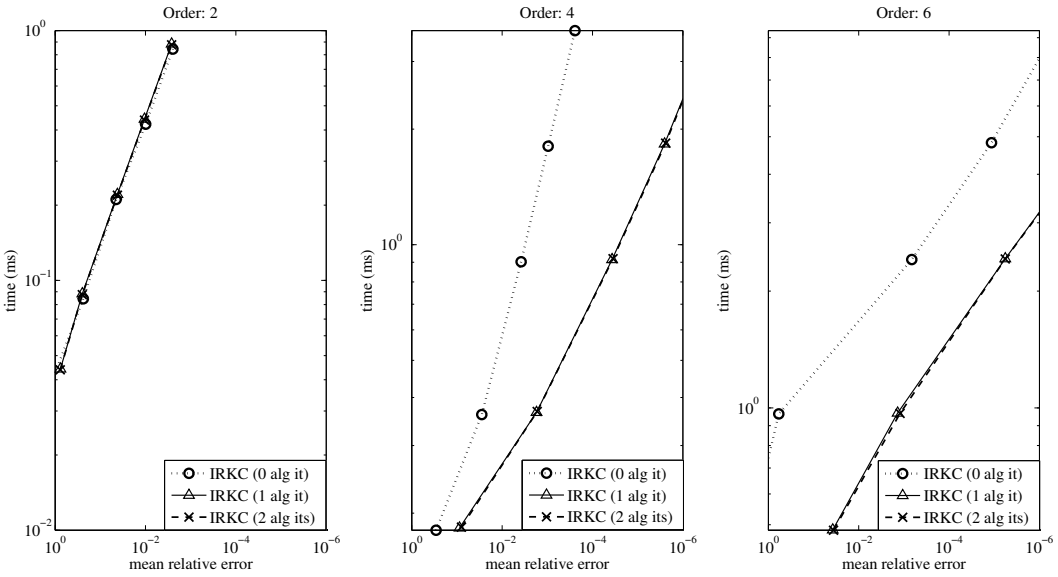



Figure 6.8: Pendulum model: The work-precision diagram for the IRKC IFT-R method to study the impact of the number of Newton iterations for the separate system of algebraic equations.

The large nonlinear system for an IRK method consists of $s(N_x + N_z)$ collocation equations, while the separate system only consists of $N_z$ algebraic equations. Since the Newton iterations are relatively cheap, it is important to always perform at

least one Newton iteration for this system of algebraic equations. The fact that one Newton iteration is sufficient confirms that the prediction from Section 4.1.2 is already quite accurate. The same conclusions are made from the results for the kite DAE model (see Section D.4).

## 6.7 Performance of the integrators

Let us now briefly look at the performance of the auto generated RK methods, comparing them to the SUNDIALS integrators where possible. SUNDIALS is a suite of nonlinear ODE and DAE solvers, written in C for either serial or parallel machine environments [34] and [54]. The relatively new state of the art solvers CVODES and IDAS will be used, which respectively handle ODE and DAE systems of equations with sensitivity analysis possibilities. Variable-order, variable-step LM and BDF methods are implemented. This actually makes these integrators less suitable for the targeted applications with tight real-time constraints. However, such methods are able to achieve a much higher accuracy with a relatively small cost in computation time. The codes of SUNDIALS are written for solving large-scale problems, while the used test systems are rather small. And for simplicity, the control inputs in the different models are assumed to be constant and equal to zero for these numerical experiments. All these aspects should be considered when making conclusions from the numerical results that are presented in this section.

### 6.7.1 Performance on ODE systems

The 3 ODE test systems from Section 6.1 are integrated over a time interval $[0, 0.1]$, representing one shooting interval. In addition to the CVODES solver of SUNDIALS, four auto generated RK methods are tested. The first one is the $4^{\text{th}}$ order ERK VDE method, which was already available in the code generation tool of ACADO. The others are the Gauss IRK IFT-R methods of order 2, 4 and 6, presented in this text. The results of these numerical experiments are shown in Table 6.3, 6.4 and 6.5 for respectively the Van der Pol oscillator, the crane and kite ODE model. These tables present the computation time and the number of integration steps performed to achieve at least a certain accuracy. They also show the achieved speedup for each accuracy, i.e. the computation time of CVODES divided by the one for the fastest auto generated RK method. The developed integrators seem to outperform CVODES by a factor of more or less 100. The speedup also seems to be much higher for lower accuracies and can be lower for higher accuracies. This is caused by the fact that the presented methods are fixed-order, fixed-step integrators suited for real-time applications while CVODES uses a variable-order, variable-step method.

| accuracy | | ERK4 | IRK2 | IRK4 | IRK6 | CVODES | speedup |
|---|---|---|---|---|---|---|---|
| 1e-1 | # steps | 11 | **4** | 3 | 2 | 17 | |
| | time | 1.7 µs | **0.7 µs** | 1.2 µs | 2.0 µs | 614 µs | 877 |
| 1e-2 | # steps | 12 | **6** | 4 | 2 | 20 | |
| | time | 1.8 µs | **1.0 µs** | 1.6 µs | 2.0 µs | 624 µs | 624 |
| 1e-3 | # steps | 12 | **7** | 5 | 2 | 25 | |
| | time | 1.8 µs | **1.2 µs** | 2.0 µs | 2.0 µs | 679 µs | 566 |
| 1e-4 | # steps | 13 | **8** | 5 | 4 | 31 | |
| | time | 2.0 µs | **1.4 µs** | 2.0 µs | 4.0 µs | 723 µs | 516 |
| 1e-5 | # steps | 14 | **11** | 6 | 5 | 43 | |
| | time | 2.1 µs | **1.9 µs** | 2.3 µs | 5.0 µs | 809 µs | 426 |
| 1e-6 | # steps | **19** | 35 | 8 | 8 | 53 | |
| | time | **2.9 µs** | 6.0 µs | 3.1 µs | 7.9 µs | 903 µs | 311 |
| | time/step | 0.15 µs | 0.17 µs | 0.39 µs | 0.99 µs | | |

Table 6.3: Integration of the Van der Pol oscillator over 0.1s.

| accuracy | | ERK4 | IRK2 | IRK4 | IRK6 | CVODES | speedup |
|---|---|---|---|---|---|---|---|
| 1e-1 | # steps | 4 | **4** | 2 | 2 | 11 | |
| | time | 44 µs | **12 µs** | 18 µs | 46 µs | 3928 µs | 327 |
| 1e-2 | # steps | 6 | 10 | **3** | 2 | 18 | |
| | time | 66 µs | 30 µs | **27 µs** | 46 µs | 4311 µs | 160 |
| 1e-3 | # steps | 8 | 30 | **4** | 3 | 29 | |
| | time | 88 µs | 90 µs | **36 µs** | 69 µs | 4859 µs | 135 |
| 1e-4 | # steps | 11 | 90 | **7** | 4 | 43 | |
| | time | 121 µs | 270 µs | **63 µs** | 92 µs | 4938 µs | 78 |
| 1e-5 | # steps | 20 | 280 | **12** | 5 | 53 | |
| | time | 220 µs | 840 µs | **108 µs** | 115 µs | 5359 µs | 50 |
| 1e-6 | # steps | 35 | 900 | 22 | **7** | 66 | |
| | time | 385 µs | 2700 µs | 198 µs | **161 µs** | 5766 µs | 36 |
| | time/step | 11 µs | 3 µs | 9 µs | 23 µs | | |

Table 6.4: Integration of the crane model over 0.1s.

| accuracy | | ERK4 | IRK2 | IRK4 | IRK6 | CVODES | speedup |
|---|---|---|---|---|---|---|---|
| 1e-1 | # steps | 1 | **1** | 1 | 1 | 15 | |
| | time | 71 µs | **15 µs** | 38 µs | 83 µs | 8215 µs | 548 |
| 1e-2 | # steps | 1 | 3 | **1** | 1 | 15 | |
| | time | 71 µs | 45 µs | **38 µs** | 83 µs | 8305 µs | 218 |
| 1e-3 | # steps | 2 | 8 | **1** | 1 | 18 | |
| | time | 142 µs | 120 µs | **38 µs** | 83 µs | 8775 µs | 231 |
| 1e-4 | # steps | 3 | 25 | **2** | 1 | 22 | |
| | time | 213 µs | 375 µs | **76 µs** | 83 µs | 8892 µs | 117 |
| 1e-5 | # steps | 5 | 80 | **3** | 2 | 27 | |
| | time | 355 µs | 1200 µs | **114 µs** | 166 µs | 9301 µs | 82 |
| 1e-6 | # steps | 8 | 250 | 5 | **2** | 33 | |
| | time | 568 µs | 3750 µs | 190 µs | **166 µs** | 9514 µs | 57 |
| | time/step | 71 µs | 15 µs | 38 µs | 83 µs | | |

Table 6.5: Integration of the kite ODE model over 0.1s.

### 6.7.2 Performance on DAE systems

Very similar experiments can be executed for the 2 DAE test systems, making the comparison with the IDAS solver of SUNDIALS. Here only the Gauss IRK IFT-R methods of order 2, 4 and 6 are tested, which also support DAE systems (see Chapter 4). And the systems will now be integrated over an interval of 1s. The results are shown in Table 6.6 and 6.7 for respectively the pendulum and kite DAE model. The conclusions from these tables are similar to the ones in the previous section. The auto generated RK methods also seem to outperform IDAS by a factor of more or less 100.

| accuracy | | IRK2 | IRK4 | IRK6 | IDAS | speedup |
|---|---|---|---|---|---|---|
| 1e-1 | # steps | **3** | 1 | 1 | 14 | |
| | time | **12 µs** | 17 µs | 49 µs | 7308 µs | 609 |
| 1e-2 | # steps | 10 | **2** | 1 | 17 | |
| | time | 40 µs | **34 µs** | 49 µs | 7890 µs | 232 |
| 1e-3 | # steps | 30 | **2** | 3 | 25 | |
| | time | 120 µs | **34 µs** | 147 µs | 8026 µs | 236 |
| 1e-4 | # steps | 100 | **4** | 4 | 42 | |
| | time | 400 µs | **68 µs** | 196 µs | 8290 µs | 122 |
| 1e-5 | # steps | 300 | **8** | 5 | 46 | |
| | time | 1200 µs | **136 µs** | 245 µs | 8374 µs | 62 |
| 1e-6 | # steps | 950 | **14** | 6 | 80 | |
| | time | 3800 µs | **238 µs** | 294 µs | 11488 µs | 48 |
| | time/step | 4 µs | 17 µs | 49 µs | | |

Table 6.6: Integration of the pendulum over 1s.

| accuracy | | IRK2 | IRK4 | IRK6 | IDAS | speedup |
|---|---|---|---|---|---|---|
| 1e-1 | # steps | **2** | 1 | 1 | 47 | |
| | time | **78 µs** | 179 µs | 434 µs | 34616 µs | 444 |
| 1e-2 | # steps | **4** | 1 | 1 | 50 | |
| | time | **156 µs** | 179 µs | 434 µs | 35977 µs | 231 |
| 1e-3 | # steps | 12 | **2** | 2 | 63 | |
| | time | 468 µs | **358 µs** | 868 µs | 39459 µs | 110 |
| 1e-4 | # steps | 35 | **3** | 3 | 69 | |
| | time | 1365 µs | **537 µs** | 1302 µs | 41317 µs | 77 |
| 1e-5 | # steps | 115 | **5** | 4 | 74 | |
| | time | 4485 µs | **895 µs** | 1736 µs | 42082 µs | 47 |
| 1e-6 | # steps | 350 | **9** | 5 | 79 | |
| | time | 13650 µs | **1611 µs** | 2170 µs | 44277 µs | 27 |
| | time/step | 39 µs | 179 µs | 434 µs | | |

Table 6.7: Integration of the kite DAE model over 1s.

### 6.7.3 Continuous Output

Let us now have a closer look at the continuous output aspect of the auto generated IRK methods. As an example, the kite DAE model from Section 6.2.2 will again be integrated over an interval of 1s. The extra assumption here is that the position of the kite $(x, y, z)$ forms an output that is needed at a frequency of 1kHz. Instead of limiting the step size of the used integrator by 0.001s or smaller, the continuous output of the presented methods will be used. In addition to the Gauss methods (IRK2, IRK4 and IRK6), the Radau IIA methods (RAD1, RAD3 and RAD5) will be used in this numerical experiment [33]. Table 6.8 presents the order of accuracy and the order of the continuous output for both these methods with 1, 2 and 3 stages. Note that the 1-stage Radau IIA method is simply the implicit Euler method.

| | | RAD1 | IRK2 | RAD3 | IRK4 | RAD5 | IRK6 |
|---|---|---|---|---|---|---|---|
| number stages | $s$ | 1 | 1 | 2 | 2 | 3 | 3 |
| order method | $p$ | 1 | 2 | 3 | 4 | 5 | 6 |
| order output | $p^*$ | 2 | 2 | 3 | 3 | 4 | 4 |

Table 6.8: Order of accuracy and of the continuous output for the 1-, 2- and 3-stage Gauss-Legendre and Radau IIA methods.

The numerical results of these experiments are presented in Table 6.9. Each group of rows shows the number of integration steps, the mean error on the states, the mean error on the outputs and the total computation time for the integration with generation of the sensitivities and the continuous output. The number of integration steps is chosen for each method to achieve a similar accuracy for the states. No more than 1000 integration steps can however be performed over 1s if the continuous output is desired to be used for the position measurements at 1kHz. This limit is for example reached for the Radau IIA method of order 1. Note that comparing the error on the states with the error on the outputs is not completely fair. The error on the outputs is namely computed as the mean error over the 1000 equidistant grid points in the interval of 1s, while the error on the states is computed for the values at the end of the interval. The error on the outputs is therefore often more pessimistic due to some outliers. Table 6.9 also shows a reference time, which is the computation time needed to integrate the system over 1s using a step size of 0.001s. This would be the way to achieve the kite position at a frequency of 1kHz if no continuous output is available. After comparing the computation times with their corresponding reference times, it should be clear that the auto generated methods provide a relatively cheap way to compute higher frequency outputs with their sensitivities.

| | RAD1 | IRK2 | RAD3 | IRK4 | RAD5 | IRK6 |
|---|---|---|---|---|---|---|
| # steps | 10 | 2 | 1 | 1 | 1 | 1 |
| error states | 7.7e-2 | 4.1e-2 | 1.5e-2 | 5.4e-3 | 5.2e-3 | 5.1e-3 |
| error outputs | 1.2e+0 | 3.0e+0 | 4.8e-1 | 7.5e-2 | 1.8e-1 | 8.9e-2 |
| time | 2490 µs | 1740 µs | 1380 µs | 1370 µs | 4500 µs | 4340 µs |
| # steps | 125 | 4 | 2 | 1 | 1 | 1 |
| error states | 6.5e-3 | 8.0e-3 | 1.8e-3 | 5.4e-3 | 5.2e-3 | 5.1e-3 |
| error outputs | 7.5e-2 | 1.3e+0 | 4.3e-2 | 7.5e-2 | 1.8e-1 | 8.9e-2 |
| time | 7400 µs | 2416 µs | 2180 µs | 1370 µs | 4500 µs | 4340 µs |
| # steps | 1000 | 20 | 4 | 2 | 2 | 2 |
| error states | 8.2e-4 | 3.0e-4 | 2.8e-4 | 4.7e-4 | 1.0e-4 | 2.8e-4 |
| error outputs | 7.5e-3 | 1.8e-1 | 2.5e-2 | 3.6e-2 | 2.0e-2 | 9.5e-3 |
| time | 43300 µs | 2960 µs | 2860 µs | 2180 µs | 5520 µs | 5540 µs |
| # steps | | 40 | 10 | 4 | 2 | 4 |
| error states | | 7.6e-5 | 1.8e-5 | 2.3e-5 | 1.0e-4 | 1.7e-6 |
| error outputs | | 7.5e-2 | 4.5e-3 | 1.3e-2 | 2.0e-2 | 9.5e-4 |
| time | | 3792 µs | 4100 µs | 2820 µs | 5520 µs | 6800 µs |
| # steps | | 125 | 20 | 5 | 4 | 4 |
| error states | | 7.8e-6 | 2.2e-6 | 8.1e-6 | 7.6e-7 | 1.7e-6 |
| error outputs | | 1.5e-2 | 1.0e-3 | 8.8e-3 | 2.1e-3 | 9.5e-4 |
| time | | 7775 µs | 5980 µs | 3075 µs | 6720 µs | 6800 µs |
| # steps | | 500 | 40 | 10 | 4 | 5 |
| error states | | 4.9e-7 | 2.8e-7 | 5.0e-7 | 7.6e-7 | 4.3e-7 |
| error outputs | | 1.2e-3 | 2.1e-4 | 2.1e-3 | 2.1e-3 | 4.5e-4 |
| time | | 22700 µs | 9640 µs | 4070 µs | 6720 µs | 7200 µs |
| reference time | 39000 µs | 39000 µs | 179000 µs | 179000 µs | 434000 µs | 434000 µs |

Table 6.9: Integration of the kite DAE model over 1s with extra outputs at 1kHz.

The results from Table 6.9 are also summarized in the work-precision diagram of Figure 6.9. This figure shows the precision of the outputs computed by the different auto generated methods, and the corresponding total computation time. Although the Radau IIA methods are stiffly accurate, the Gauss methods seem to compute the needed outputs more efficiently for this model.

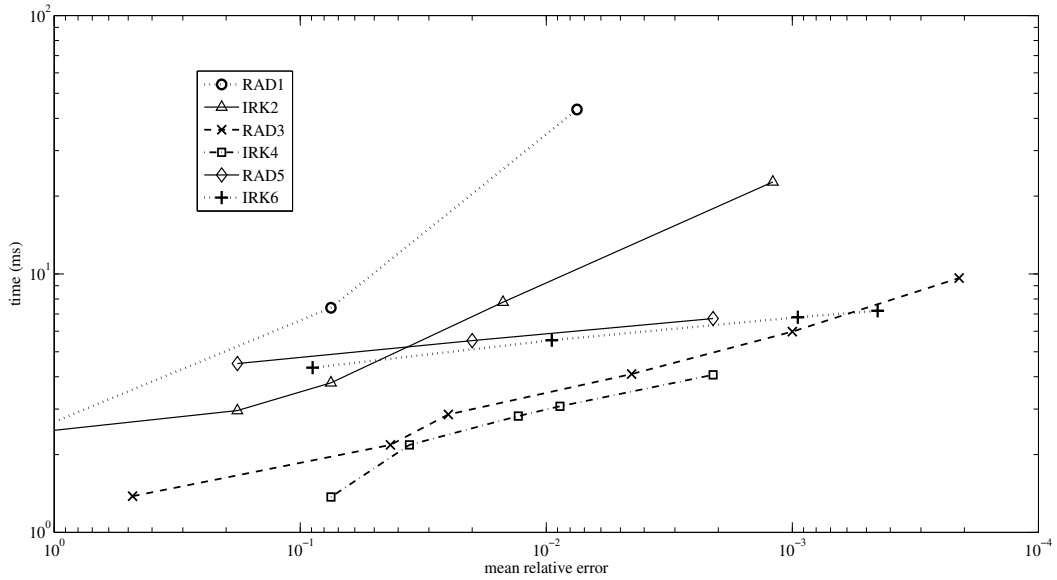



Figure 6.9: The work-precision diagram comparing different methods for the integration of the kite DAE model with extra outputs at 1kHz. It presents the total computation time with respect to the mean relative error of the outputs.

## 6.8 Impact on Embedded Optimization

This section will briefly illustrate the usage of the auto generated RK methods with the NMPC export of the code generation tool described in Section 1.5.1, similar to what is done in [58]. The crane model from Section 6.1.2 is used and the following simple OCP formulation is used:

$$\begin{aligned}
\underset{x(t),u(t)}{\text{minimize}} \quad & \int_{t_0}^{t_0+t_p} \|x(t) - x^{\text{ref}}(t)\|_Q^2 \; + \; \|u(t) - u^{\text{ref}}(t)\|_R^2 \, \mathrm{d}t \\
& + \|x(t_0 + t_p) - x^{\text{ref}}(t_0 + t_p)\|_P^2 \\
\text{subject to} \quad & x(t_0) = x_0, \\
& \dot{x}(t) = f(x(t), u(t)), \qquad \forall t \in [t_0, t_0 + t_p], \\
& \underline{u}(t) \leq u(t) \leq \bar{u}(t), \qquad \forall t \in [t_0, t_0 + t_p], \\
& \underline{x}(t) \leq x(t) \leq \bar{x}(t), \qquad \forall t \in [t_0, t_0 + t_p].
\end{aligned} \tag{6.11}$$

The OCP has a least-squares tracking objective, and 10 control intervals over a horizon of 1s are used. Table 6.10 shows timing results for NMPC of the overhead crane using single shooting. The table shows average execution times for one real-time iteration and this for the ERK method of order 4 and for the IRK Gauss methods of order 2, 4 and 6. The same OCP formulation is always used but the step size is chosen to make each integrator achieve an accuracy of about $10^{-3}$. The IRK method

of order 4 seems the most suitable here. Similar results can be expected when using the presented methods in combination with the export of MHE code.

| | ERK4 | IRK2 | IRK4 | IRK6 |
|---|---|---|---|---|
| integration step size | 0.025 s | 0.01 s | 0.05 s | 0.1 s |
| integration method | 290 µs | 266 µs | 161 µs | 222 µs |
| condensing | 154 µs | 131 µs | 142 µs | 142 µs |
| QP solution (qpOASES) | 9 µs | 9 µs | 9 µs | 10 µs |
| remaining operations | 17 µs | 14 µs | 8 µs | 6 µs |
| one real-time iteration | 470 µs | 420 µs | **320 µs** | 380 µs |

Table 6.10: Average computation times for NMPC of an overhead crane.

# Chapter 7

# Conclusion

This thesis has proposed code generation for collocation methods with efficient sensitivity generation. A detailed discussion on RK methods, the special case of collocation methods and their interesting properties has been given. The text also addressed how to efficiently implement such methods. The major advantages of using collocation methods are their stability properties for stiff systems, their possibility to handle DAE systems in a similar way and the option to easily generate continuous output. These three aspects have all been described in different chapters and have been fully exploited in the eventual implementation. In addition to integrating the system of equations, it is essential to also compute the sensitivities of the results so that they can be used within an optimizer. Different ways to generate these sensitivities have been presented from which the IFT-R method could be identified as the most efficient method. Eventually, many numerical experiments provided guidelines to set the different parameters in the implementation which has also led to default values for these parameters. Other simulations confirmed assumptions made earlier in the text or showed that it is possible for auto generated IRK methods to outperform a simple explicit method. For the targeted problems which are rather small-scale, the presented methods even can be $10 - 1000$ times faster than the state of the art solvers from SUNDIALS.

The presented code generation of IRK methods for ODE/DAE systems with efficient sensitivity generation and the continuous output feature is made available as open source code in the ACADO Toolkit. The results of this thesis initiated the support of optimal control for DAE systems and of MHE with multi-rate measurements by the ACADO code generation tool in the near future. Both of these features will for example be already useful in experiments of the ERC Highwind project in which IMU measurements will be available at a high frequency while GPS coordinates will become available at a relatively low frequency. At this instant, the auto generated IRK methods already improved the support for stiff systems. This is for example useful for the crane setup available at the KU Leuven, for which the model consists of a stiff ODE system. They have also already been successfully used in test experiments of the Highwind project. The developed methods have proven to

often outperform auto generated explicit methods and they also outperformed the CVODES and IDAS solvers from SUNDIALS. Except for applications within the KU Leuven, there will also turn up possible applications elsewhere. For example, there was already interest from the company Xsens to use the results from this thesis project in a two link test estimation problem. This application has also driven the further development of the methods.

Future work will expand the proposed methods by improving the support for larger problems. In the case of a DAE system, the sparsity structure present in the linear systems can for example be exploited when this is beneficial. Also the option of code generation for semi-implicit RK methods with efficient sensitivity generation will be explored.

# Appendices

# Appendix A

# NMPC paper

This appendix contains the paper accepted for the IFAC Conference on Nonlinear Model Predictive Control 2012 (NMPC'12). The conference takes place in Noordwijkerhout, the Netherlands on August 23 – 27, 2012.

# Auto Generation of Implicit Integrators for Embedded NMPC with Microsecond Sampling Times [1]

**Rien Quirynen, Milan Vukov and Moritz Diehl**

**Abstract:** Algorithms for fast real-time Nonlinear Model Predictive Control (NMPC) for mechatronic systems face several challenges. They need to respect tight real-time constraints and need to run on embedded control hardware with limited computing power and memory. A combination of efficient online algorithms and code generation of explicit integrators was shown to be able to overcome these hurdles. This paper generalizes the idea of code generation to Implicit Runge-Kutta (IRK) methods with efficient sensitivity generation. It is shown that they often outperform existing auto-generated Explicit Runge-Kutta (ERK) methods. Moreover, the new methods allow to treat Differential Algebraic Equation (DAE) systems by NMPC with microsecond sampling times.

## 1. INTRODUCTION

Nonlinear Model Predictive Control (NMPC) has become a quite popular approach to achieve optimal control and the techniques are well established [Mayne et al., 2000], [Simon et al., 2009]. The use of Nonlinear MPC for real-time control of a system is a relatively easy step for systems with slow dynamics. It is however still a major challenge for many mechatronic systems. A model is needed which describes the system well without being too complex. The used numerical algorithms need to compute the new optimal controls satisfying the hard real-time constraints. Concerning the latter topic, a lot of progress has been made quite recently which is for example described in [Jones and Morari, 2010], [Kirches et al., 2010] and [Diehl et al., 2009].

To minimize the computational delay for the solution of the next Optimal Control Problem (OCP), a combination of efficient online algorithms and code optimization is needed. Such online algorithms are based on principles such as offline precomputation, delay compensation by prediction or division of the computations in a preparation and a feedback phase. The preparation phase is typically the most CPU intensive one, while the feedback phase can quickly deliver an approximate solution to the OCP. The Real-Time Iteration (RTI) scheme is such an online algorithm for NMPC which performs only one SQP type iteration with Gauss-Newton Hessian per sampling time and it uses this division in two phases [Diehl et al., 2002]. This scheme is implemented in the ACADO Toolkit [Houska et al., 2011].

In addition to using efficient and adapted algorithms, code optimization is necessary which is possible in the form of code generation. Quite recently, code generation has attracted great attention due to the software package *CVXGEN* which applies this idea to real-time convex optimization [Mattingley and Boyd, 2009]. Real-time optimization algorithms are often run on embedded hardware imposing extra restrictions concerning the computing power and memory. The reasons to use code generation are the speed-up that can be achieved by removing unnecessary computations, the optimized memory management because problem dimensions and sparsity patterns are known and the more efficient cache usage by reorganizing the computations.

This also motivated the development of the ACADO code generation tool which exports highly efficient, self-contained C-code solving a specific optimal control problem [Ferreau, 2011]. The RTI algorithm is based on a shooting discretization of the OCP. The integration of the system of Ordinary Differential Equations (ODE) or Differential Algebraic Equations (DAE) and the computation of the sensitivities are a major computational step. In order to guarantee a deterministic runtime, a fixed step integration method is preferable here. The tool previously only exported an Explicit Runge-Kutta (ERK) method of order $4$ so there was a limited applicability for stiff systems and no support for DAE systems.

*Contribution*: This paper proposes code generation for an efficient implementation of an Implicit Runge-Kutta (IRK) method with sensitivity generation and shows that it outperforms code generation of an ERK method in the case of a stiff ODE system and that it can efficiently handle index-1 DAE systems.

The paper is organized as follows. Section 2 describes Runge-Kutta (RK) methods with a focus on the IRK methods. Section 3 proposes different ways to efficiently compute the sensitivities. Section 4 shows the results of numerical experiments that illustrate the performance of the described implementation on ODE and DAE systems. Section 5 finally discusses the integration in the ACADO Toolkit.

## 2. IMPLICIT RUNGE-KUTTA METHODS (IRK)

This section presents a discussion on IRK methods. The formulation and some properties of the RK methods are treated in Subsection 2.1 and 2.2, while Subsection 2.3 describes some implementation details. Subsection 2.4 presents the extension to index-1 DAE systems.

### *2.1 Formulation*

As a subtask in shooting methods for dynamic optimization, the following Initial Value Problem (IVP) needs to be solved over a time interval:

[1] R. Quirynen, M. Vukov and M. Diehl are with the Optimization in Engineering Center (OPTEC), K.U. Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium. rien.quirynen@student.kuleuven.be

$$\begin{aligned} \dot{x}(t) &= f(t, x(t)), \\ x(0) &= x_0, \end{aligned} \tag{1}$$

with $x(t)$ a vector of $N_x$ differential states. In what follows, $x_j$ will denote the approximation of the solution $x(t)$ at time point $t_j$.

Unlike multistep methods, single-step methods refer to only one previous solution point and its derivative to determine the current value. An $s$-stage RK method has $s$ coefficients $c_i$ and $s$ stage values $X_i$ which can be viewed as an approximate solution at time $t_n + c_i h$ with $t_n$ the current time instant. In accordance with [Hairer et al., 1986], the variables $k_i = f(t_n + c_i h, X_i)$ will be used. This results in the following formulation for a RK method with $s$ stages:

$$\begin{cases} k_1 = f(t_n + c_1 h, x_n + h \sum_{j=1}^{s} a_{1j} k_j), \\ \quad \cdots \\ k_s = f(t_n + c_s h, x_n + h \sum_{j=1}^{s} a_{sj} k_j), \\ x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i, \end{cases} \tag{2}$$

where $b_i$ are the weights and $a_{ij}$ are the internal coefficients. The internal consistency conditions $c_i = \sum_{j=1}^{s} a_{ij}$ are typically satisfied. For the method to have at least order 1, the weights must obey $\sum_{i=1}^{s} b_i = 1$. Any RK method that satisfies this first order condition is consistent and therefore also convergent [Frank, 2008]. The formulas in (2) are often represented using a Butcher table:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \equiv \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{3}$$

For an ERK method the matrix $A$ is strictly lower triangular which means that the variables $k_i$ are defined explicitly in (2). The focus of this paper is however directed more towards the IRK methods which do not have these restrictions.

### 2.2 Properties

There are two important advantages of the implicit methods which should be mentioned here. There always exists an IRK method with $s$ stages that has an order of accuracy equal to $2s$ [Hairer et al., 1986]. However, there is no ERK method of $s$ stages that has an order larger than $s$. The first advantage of the implicit methods is therefore their generally high order of accuracy.

The second advantage is related to stability properties for integration of stiff systems. An integration method with a finite region of absolute stability needs a step size which is excessively small with respect to the smoothness of the exact solution for the time interval on which a system is stiff. In general, IRK methods have a larger region of absolute stability than ERK methods, making them more suited for stiff systems. A method whose region of absolute stability contains the left complex half-plane $\mathbb{C}^-$, is called A-stable. In this paper, we are most interested in A-stable IRK methods such as the Gauss-Legendre or Radau methods [Hairer and Wanner, 1991].

### 2.3 Implementation

An IRK method needs to solve the system of $N_x \times s$ nonlinear equations in (2). This is typically done using the Newton method or a variant of this method where the Jacobian (or its inverse) is approximated. The system is summarized as:

$$\begin{cases} F(w_n, k) = 0, \\ \quad x_{n+1} = \phi(x_n, k), \end{cases} \tag{4}$$

where $k = (k_1, \ldots, k_s)$ and $w_n = (x_n, u_n)$ with $x_n$ and $u_n$ the current values for the states and control inputs. A variant of the Newton method to solve the nonlinear system $F(w_n, k) = 0$ can be presented as

$$\begin{aligned} M &= \frac{\partial F}{\partial k}(w_n, k^{[0]}) \\ k^{[i]} &= k^{[i-1]} - M^{-1} F(w_n, k^{[i-1]}),\ i = 1, \ldots, L \end{aligned} \tag{5}$$

The assumption here is that a fixed number of $L$ Newton iterations is performed and that the initial guess $k^{[0]}$ for the variables $k = (k_1, \ldots, k_s)$ is sufficiently good. In real-time applications, the algorithm execution time is strictly constrained and it is desirable to avoid conditional statements in the exported code. Therefore, the step size and the number of Newton iterations are fixed. The second assumption means that a few Newton iterations are sufficient and that the successive values of the Jacobian do not differ much, which justifies the fact that the Jacobian $\partial F / \partial k$ is evaluated only once here. The $s$ different evaluations of the Jacobian $\partial f / \partial x$ can preferably be done using Algorithmic Differentiation (AD) [Griewank and Walther, 2008].

For the dense linear system $M \Delta k^{[i]} = -F(w_n, k^{[i-1]})$ with $\Delta k^{[i]} = k^{[i]} - k^{[i-1]}$, two customized versions of a linear solver are exported. Equation (5) shows that the same linear system needs to be solved multiple times with a different right-hand side. The first version of the exported linear solver therefore computes the factorization of the matrix $M$ and solves the system, while the second version can reuse this factorization of the matrix $M$. This way, the factorization of the matrix $M$ will only be computed once.

The LU factorization using Gaussian elimination will be used here, but also the QR factorization using Householder triangularization is possible. The latter however requires $\sim \frac{4}{3} n^3$ flops for an $n \times n$ matrix, while Gaussian elimination requires only $\sim \frac{2}{3} n^3$ flops [Golub and Loan, 1996]. After the LU factorization, only an upper-triangular system needs to be solved which requires $\sim n^2$ flops using back substitution. This means that the second version of the exported linear solver only needs to transform the new right-hand side and then to perform the same back substitution which requires only $\sim 2n^2$ flops in total.

The initialization of the "$k$-variables" is important and addressed as follows. In the first integration step, no previous information is available which means that we initialize $k$ with zero and that some extra Newton iterations in (5) will be performed. That is an efficient way to handle this, since the factorization of the matrix $M$ is reused in every iteration. An alternative solution would be to use an ERK method to initialize the variables in the first integration step. It is useful to clarify what is meant here with the *first* integration step. In the case of single shooting, this is the first integration step in the first

shooting interval. In the case of multiple shooting, this is the first integration step in every shooting interval.

For the initialization of the variables in a subsequent step, the converged values from the previous step are available. A first possibility is to simply use these converged values as the initial guess for the next integration step. This is the most practical initialization and can already work quite well. There is another possibility in the case of collocation methods. Collocation methods are a special case of IRK methods and they provide a continuous approximation of the solution $x(t)$ on each interval $[t_n, t_{n+1}]$ [Frank, 2008]. This collocation polynomial can be extrapolated to obtain an initialization of the variables $k$ for the next integration step, which can often be more accurate. In what follows, the discussed IRK methods are collocation methods and the implementation uses the latter prediction method which was also proposed in [Cong and Xuan, 2003].

### *2.4 Differential-Algebraic equations*

Let $z$ contain the $N_z$ algebraic states which are defined by the algebraic equations $0 = g(t, x, z)$. The assumption here is that the DAE system is of index 1 which means that the matrix $\partial g/\partial z$ is invertible [Findeisen and Allgöwer, 2000]. This paper will assume the following implicit DAE formulation:

$$\begin{aligned} 0 &= f(t, \dot{x}(t), x(t), z(t)) \\ 0 &= g(t, x(t), z(t)) \end{aligned} \tag{6}$$

The generalization of the IRK methods to index-1 DAE systems can be done in two ways. The easiest way is to use only the differential states $x$ as independent variables and to reduce the DAE to the implicit ODE $f(t, \dot{x}, x, z(x)) = 0$. For nonlinear algebraic equations, this is however not efficient since this requires internal iterations [Schulz et al., 1998].

The alternative way is to explicitly deal with the algebraic equations in (6) by solving the following system:

$$0 = f(t_n + c_i h, k_i, x_n + h \sum_{j=1}^{s} a_{ij} k_j, Z_i), \tag{7a}$$

$$0 = g(t_n + c_i h, x_n + h \sum_{j=1}^{s} a_{ij} k_j, Z_i), \ i = 1, \ldots, s, \tag{7b}$$

$$x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i, \tag{7c}$$

$$0 = g(t_{n+1}, x_{n+1}, z_{n+1}), \tag{7d}$$

with $Z_i$ the stage values of the algebraic states, satisfying the algebraic equations. The nonlinear system consisting of (7a) and (7b) can be solved in a similar way as described for ODE. The expression for $x_{n+1}$ is still explicit and (7d) makes sure that also the new values for the states are consistent. The value for $z_{n+1}$ can first be predicted based on $z_n$ and the values $Z_i$, such that only a few Newton iterations are needed to solve this separate nonlinear system. This will be the preferred way to deal with DAE systems. Note that for the Radau IIA methods (7d) is already contained in (7b).

## 3. SENSITIVITY GENERATION

Next to the new values of the states, the integration method also needs to deliver their sensitivities to the optimizer that handles the OCP. Let us assume in what follows that $w_n = (x_n, u_n)$ denotes all the independent variables with respect to which the first derivatives of the results are needed. Different approaches to calculate the matrix $\partial(x_{n+1}, z_{n+1})/\partial w_n$ will be shortly described and evaluated in this section. The goal is to agree on the most efficient way to be used in combination with an IRK method.

The Variational Differential Algebraic Equations (VDAE) are treated in Subsection 3.1, the principle of Internal Numerical Differentiation (IND) is discussed in Subsection 3.2 and Subsection 3.3 presents a method based on the Implicit Function Theorem (IFT).

### *3.1 Variational Differential Equations*

The VDAE can be obtained by differentiating the model equations in (6) with respect to $w_n$:

$$\begin{aligned} 0 &= \frac{\partial f}{\partial \dot{x}} \frac{\partial G_w^x}{\partial t} + \frac{\partial f}{\partial x} G_w^x + \frac{\partial f}{\partial z} G_w^z + \frac{\partial f}{\partial w_n}, \\ 0 &= \frac{\partial g}{\partial x} G_w^x + \frac{\partial g}{\partial z} G_w^z + \frac{\partial g}{\partial w_n}, \end{aligned} \tag{8}$$

with the sensitivity matrices $G_w^x = \partial x/\partial w_n$ and $G_w^z = \partial z/\partial w_n$. The idea of this approach is to augment the system with these VDAE whose unique solution is the needed sensitivity matrix $\partial(x, z)/\partial w_n$. This would however be too expensive for an implicit integration method. It is possible to exploit the structure in the augmented system, but that would lead to a method which is more or less equivalent to the other methods. Moreover, the VDAE do not describe the sensitivities of the numerical results for the discretized system which are in fact needed by the optimizer.

### *3.2 Internal Numerical Differentiation (IND)*

The second approach is based on the fundamental principle of IND due to Bock [Bock, 1983]. It generates the needed derivatives by differentiation of the integration method itself. The major difference with External Numerical Differentiation (END) is that the adaptive parts of the method are frozen during sensitivity generation. Since there are no adaptive parts in the discussed implementation for real-time applications, there is basically no difference between IND and END here.

One way to implement this is by using Finite Differences (FD). The idea is then to integrate the nominal and disturbed trajectories using the same discretization scheme [Kirches, 2006]. Using FD is however not a very accurate way to compute the sensitivities. The same principle of IND can however also be implemented using AD. The corresponding iteration scheme can be obtained by directly differentiating the equations in (5), combined with the update formulas in (7). This results in

$$\begin{aligned} \frac{dK^{[i]}}{dw_n} &= \frac{dK^{[i-1]}}{dw_n} - M^{-1}\left(\frac{\partial G}{\partial w_n} + \frac{\partial G}{\partial K}\frac{dK^{[i-1]}}{dw_n}\right) \\ \frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial x_n}{\partial w_n} + h \sum_{i=1}^{s} b_i \frac{dk_i^{[L]}}{dw_n} \\ \frac{\partial z_{n+1}}{\partial w_n} &= -\frac{\partial g}{\partial z}^{-1}\left(\frac{\partial g}{\partial w_n} + \frac{\partial g}{\partial x}\frac{\partial x_{n+1}}{\partial w_n}\right) \end{aligned} \tag{9}$$

Note that the function $G(w_n, K)$ is defined as $G = (f, g)$, the matrix $M = \partial G/\partial K(w_n, K^{[0]})$ and the variables $K = (k, Z)$,

**Algorithm 1.** One step of the IFT-R method

**Input:** consistent $(x, z, u)_n$, initial $K^{[0]}$, LU factorization of matrices $M$ and $N$

**Output:** $(x, z)_{n+1}$ and $\partial(x, z)_{n+1}/\partial w_n$

$w_n \leftarrow (x, u)_n$
**if** $n = 0$ **then**
  $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[0]})$
  factorize $M$
  $K^{[0]} \leftarrow K^{[0]} - M^{-1}G(w_n, K^{[0]})$
**end if**
**for** $i = 1 \rightarrow L$ **do**
  $K^{[i]} \leftarrow K^{[i-1]} - M^{-1}G(w_n, K^{[i-1]})$
**end for**
$x_{n+1} \leftarrow x_n + h\sum_{i=1}^{s} b_i k_i$
predict $z_{n+1}$ using $z_n$ and $Z_i$ values
**if** $n = 0$ **then**
  $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$
  factorize $N$
**end if**
$z_{n+1} \leftarrow z_{n+1} - N^{-1}g(t_{n+1}, x_{n+1}, z_{n+1})$
$M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[L]})$ and $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$
factorize $M$ and $N$
compute sensitivities using $M$ and $N$ in (10)
initialize $K^{[0]}$ for next integration step
$n \leftarrow n + 1$

consistent with (7). This method to generate the sensitivities will be called the IND-AD method. This iterative scheme can be proven to converge to the results of the IFT method which is discussed below. The major disadvantage of the IND-AD method is that it is costly when we compute many directional derivatives, as we do here.

### *3.3 Implicit Function Theorem*

The final approach to compute the sensitivities is based on the Implicit Function Theorem (IFT) and will therefore be denoted by the term "IFT method". Applying the IFT to (7) results in

$$\begin{aligned} \frac{dK}{dw_n} &= -\frac{\partial G}{\partial K}^{-1}\frac{\partial G}{\partial w_n}, \\ \frac{\partial x_{n+1}}{\partial w_n} &= \frac{\partial x_n}{\partial w_n} + h\sum_{i=1}^{s} b_i \frac{dk_i}{dw_n}, \\ \frac{\partial z_{n+1}}{\partial w_n} &= -\frac{\partial g}{\partial z}^{-1}\left(\frac{\partial g}{\partial w_n} + \frac{\partial g}{\partial x}\frac{\partial x_{n+1}}{\partial w_n}\right), \end{aligned} \tag{10}$$

with $G = (f, g)$ and $K = (k, Z)$, consistent with (7). Note that the factorization of the matrices $\partial G/\partial K$ and $\partial g/\partial z$ only needs to be computed once. This IFT method can simply be combined with the IRK method from (7) to form an integrator with sensitivity generation. Clearly, the LU factorization is then a large cost in this implementation which is incurred twice per integration step for each of the matrices.

The Newton iterations of the IRK method can however reuse the matrix evaluation and LU factorization from the IFT in the previous integration step. The underlying assumption is that this Jacobian still serves as a good approximation. So instead of evaluating and factorizing it at the first Newton iteration as in (5), we can use the factorized Jacobian from the previous IRK step that we needed to compute for the sensitivities according to the IFT. Algorithm 1 describes this alternative implementation, further referred to as the IFT-R method. It computes the sensitivities up to machine precision without the need for an extra LU factorization, except for the first integration step.

| order IRK | FD | IND-AD | IFT | IFT-R |
|---|---|---|---|---|
| 2 | 4.8 µs | 4.8 µs | 3.0 µs | 2.2 µs |
| 4 | 15.7 µs | 23.3 µs | 12.7 µs | 8.0 µs |
| 6 | 38.0 µs | 61.8 µs | 33.8 µs | 21.2 µs |

Table 1. Average computation time per integration step for overhead crane example.

| | FD | IND-AD | IFT | IFT-R |
|---|---|---|---|---|
| LU factorization | 20.6 % | 16.1 % | 53.6 % | 44.2 % |
| linear system | 39.6 % | 27.8 % | 27.3 % | 41.9 % |
| evaluation Jacobian | 1.9 % | 3.7 % | 5.3 % | 5.1 % |
| other computations | 37.9 % | 52.4 % | 13.9 % | 8.8 % |
| total | 15.7 µs | 23.3 µs | 12.7 µs | 8.0 µs |

Table 2. Composition of the average computation time per integration step for the $4^{\text{th}}$ order Gauss method on crane model.

## 4. NUMERICAL EXPERIMENTS

This section presents numerical experiments to confirm some assumptions and to illustrate the performance of the auto generated integrators. All the numerical experiments presented in this paper are run on an ordinary computer (Intel P8600 3MB cache, 2.40 GHz, 64-bit Ubuntu 10.04 LTS with Clang 3.1) and the time measurements are done using the `gettimeofday` function. A comparison between the different methods to generate the sensitivities is made in Subsection 4.1, while Subsection 4.2 and 4.3 present the performance of the implemented solver on respectively ODE and DAE systems. Subsection 4.4 eventually discusses the computational complexity.

### *4.1 Efficient Derivative Generation*

The goal of these experiments is to compare the efficiency of the different methods, discussed in Section 3 to generate the needed sensitivities. Only the approach using the VDAE will not be included in the comparison because it does not compete with the other approaches in combination with an IRK method. As a test system, a model for an overhead crane similar to [Vukov et al., 2012] is used. It consists of a stiff ODE system with 8 states and 2 control inputs.

Table 1 shows the average computation time per integration step for the FD, IND-AD, IFT and IFT-R approach to compute the sensitivities in combination with the Gauss methods [Hairer and Wanner, 1991] of order 2, 4 and 6. The IFT-R method is the fastest one. This is also clear from Table 2 which shows for the $4^{\text{th}}$ order Gauss method the fraction of time spent in the LU factorization, the solution of a linear system reusing this factorization, evaluations of the Jacobian and other computations. The latter comprises of all the extra computations related to the IRK method or the sensitivity generation and can be considered as overhead.

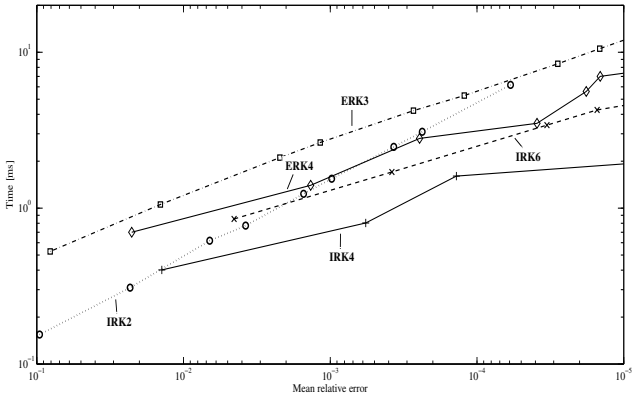



Fig. 1. An efficiency comparison of RK methods on a stiff ODE model.

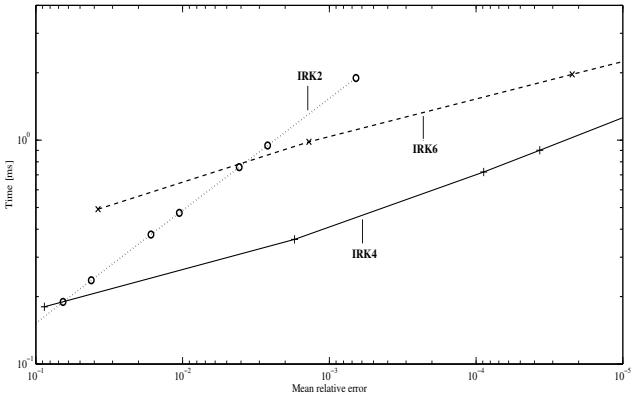



Fig. 2. An efficiency comparison of IRK methods on a DAE model.

### *4.2 Performance on stiff ODE*

The second test system is a stiff ODE describing a tethered plane using 11 differential states and 3 control inputs. It consists of a model for the plane kinematics and the tether is modeled as an elastic rod. Figure 1 presents a work-precision diagram for this test system, similar to [Hairer and Wanner, 1991]. The IRK methods are again the Gauss methods of order 2, 4 and 6, while the ERK methods are the classical method of order 4 and the 3-stage method of order 3 [Hairer et al., 1993]. The figure shows the average computation time needed to integrate the system - and to generate the desired sensitivities - over an interval of 1 second with a certain accuracy, i.e. the mean relative error. The curves for the different methods are obtained by independently choosing appropriate values for the step size. This allows us to compare the methods in a fair way.

The important conclusions from this figure are that the IRK methods are able to outperform the ERK methods for this stiff ODE system and that a higher order method can more efficiently provide a higher accuracy. The latter means that the IRK method of order 2 is preferable when a relative error of $10^{-1}$-$10^{-2}$ is sufficient. When a higher accuracy is needed, the order 4 method should be used. The IRK methods outperform the ERK methods because they can take much larger steps for the same accuracy. To indicate the speed of the auto generated integrators, the same test system is also integrated with the state of the art solver CVODES of SUNDIALS [Hindmarsh et al., 2005]. The integration with sensitivity generation then takes 100 ms, which is approximately 100 times slower than the presented auto generated IRK methods. It however needs to be said that the CVODES integrator is more suited for large-scale problems and less applicable for real-time applications since it uses a variable-order, variable-step Linear Multistep (LM) method. The computation time for CVODES is approximately independent of the required accuracy in the range which is shown here.

### *4.3 Performance on DAE*

The performance of the implemented IRK methods is illustrated on a simple DAE system describing a pendulum. The system consists of 6 differential states, 5 algebraic states and 1 control input. Figure 2 presents a work-precision diagram for the IRK methods on this test system, which is now integrated over an interval of 10 seconds. To indicate the speed of these generated integrators, the DAE test system is also integrated using the IDAS solver of SUNDIALS [Hindmarsh et al., 2005]. This takes 90 ms for the considered accuracies which again means that the presented IRK methods using code generation are approximately 100 times more efficient.

| | ERK4 | IRK2 | IRK4 | IRK6 |
|---|---|---|---|---|
| integration step size | 0.025 s | 0.01 s | 0.05 s | 0.1 s |
| integration method | 290 µs | 266 µs | 161 µs | 222 µs |
| condensing | 154 µs | 131 µs | 142 µs | 142 µs |
| QP solution (qpOASES) | 9 µs | 9 µs | 9 µs | 10 µs |
| remaining operations | 17 µs | 14 µs | 8 µs | 6 µs |
| one real-time iteration | 470 µs | 420 µs | 320 µs | 380 µs |

Table 3. Average computation times for NMPC of a crane.

### *4.4 Computational complexity*

After profiling the generated code for different test systems, one can conclude that most of the computation time is spent in the linear solver. The computational complexity of the IRK method therefore is $O(s^3(N_x + N_z)^3)$. This work might be reduced by exploiting the sparsity present in the linear systems.

## 5. INTEGRATION IN ACADO TOOLKIT

The IRK methods presented in this paper are available within the ACADO code generation tool. It is possible to export the integrators within NMPC and Moving Horizon Estimation (MHE), or as standalone components.

To illustrate the impact of using an IRK method in the case of a stiff system, Table 3 shows results for NMPC of a crane with a model similar to one presented in [Vukov et al., 2012]. The table shows average execution times for one real-time iteration and this for the ERK method of order 4 and for the IRK Gauss methods of order 2, 4 and 6. The same OCP formulation is used but the step size is chosen to make each integrator achieve an accuracy of about $10^{-3}$. The IRK method of order 4 seems the most suitable here.

## 6. CONCLUSION & FURTHER DEVELOPMENTS

This paper has proposed code generation for IRK methods with efficient sensitivity generation. A detailed discussion on the formulation and implementation of IRK methods and their properties has been given. The paper also described a natural extension of this scheme to deal with algebraic equations. Different ways to compute the sensitivities are presented from which the most efficient method could be identified. Numerical experiments confirmed that the presented implementation makes it possible for auto generated IRK methods to outperform a simple explicit method. For the small-scale problems targeted, the presented methods also seem approximately 100 times faster than the integrators from SUNDIALS. The methods are available in the ACADO Toolkit as open source code.

Future work will expand the proposed scheme by improving the support for larger problems and partially exploiting the sparsity present in the linear systems.

## ACKNOWLEDGEMENTS

The authors want to thank Hans Joachim Ferreau, Mario Zanon and Sébastien Gros for inspiring discussions and for providing test models. This research was financially supported by Research Council KUL: GOA/10/09 MaNet , Optimization in Engineering Center (OPTEC) PFV/10/002, IOF-SCORES4CHEM, FWO projects: G0226.06 (cooperative systems and optimization), G0321.06 (Tensors), G.0302.07 (SVM/ Kernel), G.0320.08 (convex MPC), G.0558.08 (Robust MHE), G.0557.08 (Glycemia2), G.0588.09 (Brain-machine) research communities (WOG: ICCoS, ANMMM, MLDM); G.0377.09 (Mechatronics MPC) IWT: PhD Grants, Eureka-Flite+, SBO LeCoPro, SBO Climaqs, SBO POM, O&O-Dsquare Belgian Federal Science Policy Office: IUAP P6/04 (DYSCO, Dynamical systems, control and optimization, 2007-2011); EU: ERNSI; FP7-HD-MPC (INFSO - ICT - 223854), COST intelliCIS, FP7-EMBOCON (ICT- 248940), FP7-SADCO ( MC ITN-264735), ERC HIGHWIND (259 166). Contract Research: AMINAL, ACCM

## REFERENCES

H.G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser, Boston, 1983.

Nguyen Huu Cong and Le Ngoc Xuan. *Parallel-Iterated RK-type PC Methods with Continuous Output Formulas*. *International Journal of Computer Mathematics*, 80:8:1025–1035, 2003.

M. Diehl, H. J. Ferreau, and N. Haverbeke. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pages 391–417. Springer, 2009.

Moritz Diehl, H. Georg Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. *Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations*. *Journal of Process Control*, 12:577–585, 2002.

H.J. Ferreau. *Model Predictive Control Algorithms for Applications with Millisecond Timescales*. PhD thesis, K.U. Leuven, 2011.

R. Findeisen and F. Allgöwer. Nonlinear model predictive control for index–one DAE systems. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, pages 145–162, Basel Boston Berlin, 2000. Birkhäuser.

Jason Frank. Numerical modelling of dynamical systems. Lecture notes, URL: `http://homepages.cwi.nl/~jason/Classes/numwisk/index.html`, 2008.

G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives:* Principles and Techniques of Algorithmic Differentiation*, Second Edition*. SIAM, 2008. ISBN 978-0-898716-59-7.

E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II:* Stiff and Differential-Algebraic Problems. Springer-Verlag, 1991. ISBN 3-540-53775-9.

E. Hairer, Syvert P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I:* Nonstiff Problems. Springer, 1986. ISBN 978-3-540-56670-0.

E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I.* Springer Series in Computational Mathematics. Springer, Berlin, 2nd edition, 1993.

A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31:363–396, 2005.

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*, 32(3):298–312, 2011. doi: 10.1002/oca.939. URL `http://onlinelibrary.wiley.com/doi/10.1002/oca.939/abstract`.

C.N. Jones and M. Morari. Polytopic approximation of explicit model predictive controllers. *IEEE Transactions on Automatic Control*, 55(11):2542–2553, 2010. ISSN 0018-9286.

C. Kirches. A Numerical Method for Nonlinear Robust Optimal Control with Implicit Discontinuities and an Application to Powertrain Oscillations. Diploma thesis, University of Heidelberg, October 2006.

C. Kirches, L. Wirsching, S. Sager, and H.G Bock. Efficient numerics for nonlinear model predictive control. In M. Diehl, Francois F. Glineur, and E. Jarlebring W. Michiels, editors, *Recent Advances in Optimization and its Applications in Engineering*, pages 339–357. Springer, 2010. doi: 10.1007/978-3-642-12598-0_30. URL `http://www.springerlink.com/content/u1550q4171540148/`.

J. Mattingley and S. Boyd. *Convex Optimization in Signal Processing and Communications*, chapter Automatic Code Generation for Real-Time Convex Optimization. Cambridge University Press, 2009.

D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: stability and optimality. *Automatica*, 26(6):789–814, 2000.

V.H. Schulz, H.G. Bock, and M.C. Steinbach. Exploiting invariants in the numerical solution of multipoint boundary value problems for DAEs. *SIAM Journal on Scientific Computing*, 19:440–467, 1998.

L.L. Simon, Z.K. Nagy, and K. Hungerbuehler. *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Swelling Constrained Control of an Industrial Batch Reactor Using a Dedicated NMPC Environment: OptCon, pages 531–539. Springer, 2009.

M. Vukov, W. Van Loock, B. Houska, H.J. Ferreau, J. Swevers, and M. Diehl. Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation. In *The 2012 American Control Conference, Montreal, Canada.*, 2012. (accepted).

# Appendix B

# Poster

This appendix contains a poster summarizing the results of this project for the master thesis fair.

# Runge-Kutta integrators with continuous output for ultra fast Moving Horizon Estimation


KATHOLIEKE UNIVERSITEIT LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Master Wiskundige ingenieurstechnieken

Masterproef *Rien Quirynen*

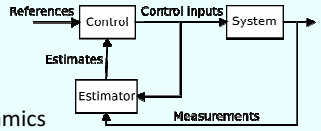
Promotor *Moritz Diehl*

Begeleider *Milan Vukov*

Academiejaar 2011-2012


## Context

**Challenge:** real-time optimal control and estimation




- fast dynamics
- tight real-time constraints
- embedded hardware

**Approach:** combination of

- efficient algorithms
- optimization and customization

code generation tool in ACADO

## Motivation project

- Implicit Runge-Kutta (IRK) methods for stiff systems
- efficient sensitivity generation
- Differential Algebraic Equations (DAE)

$$0 = f(t, \dot{x}(t), x(t), z(t))$$
$$0 = g(t, x(t), z(t))$$

- Moving Horizon Estimation (MHE) with multi-rate measurements

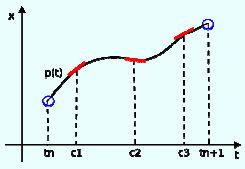
collocation methods

## Collocation methods

- IRK methods

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

- collocation polynomial




**Implementation:**

- Newton variant for nonlinear system
- Algorithmic Differentiation (AD)

$$0 = f(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij} k_j, Z_i)$$
$$0 = g(t_n + c_i h, x_n + h\sum_{j=1}^{s} a_{ij} k_j, Z_i)$$
$$x_{n+1} = x_n + h\sum_{i=1}^{s} b_i k_i$$
$$0 = g(t_{n+1}, x_{n+1}, Z_{n+1})$$

## Sensitivity generation

Shooting methods need sensitivities

**Approaches:**

- Variational DAE
- Internal Numerical Differentiation (IND) using Finite Differences (FD) or AD (IND-AD)
- Implicit Function Theorem (IFT)

**IFT-R method:**

- Gauss LU decomposition
- IFT for sensitivities
- Newton with reuse of Jacobian from IFT in previous step

## Results

One step of Gauss IRK4 for crane:

| | FD | IND-AD | IFT | IFT-R |
|---|---|---|---|---|
| LU | 20.6 % | 16.1 % | 53.6 % | 44.2 % |
| linear system | 39.6 % | 27.8 % | 27.3 % | 41.9 % |
| jacobian | 1.9 % | 3.7 % | 5.3 % | 5.1 % |
| other | 37.9 % | 52.4 % | 13.9 % | 8.8 % |
| | 18.2 µs | 24.2 µs | 13.7 µs | **9.0 µs** |

One iteration NMPC of crane:



| | ERK4 | IRK2 | IRK4 | IRK6 |
|---|---|---|---|---|
| step size | 0.025 s | 0.01 s | 0.05 s | 0.1 s |
| integration | **290 µs** | **280 µs** | **181 µs** | **243 µs** |
| condensing | 154 µs | 131 µs | 142 µs | 142 µs |
| QP | 9 µs | 9 µs | 9 µs | 10 µs |
| other | 17 µs | 14 µs | 8 µs | 6 µs |
| | 470 µs | 434 µs | **340 µs** | 401 µs |

## Achievements

- improved support stiff systems
- added support DAE systems
- allowed MHE with multi-rate measurements
- outperformed Sundials integrators by factor of **10-1000**

# Appendix C

# Software implementation

This appendix presents a compact description of the C++ software implementation within the ACADO Toolkit. The goal is not to discuss the complete implementation of the ACADO Toolkit, but to point out the code contributions of this project and potentially to clarify some connections with existing classes. Figure C.1 provides an inheritance diagram consisting of newly added classes (in bold font) and classes which are directly related. The solid lines in this figure correspond to a generalization relationship or inheritance, while the dashed line corresponds to a dependency. The figure also indicates the structure in which the software implementation will be discussed. First of all, the export of integration methods will be discussed in Section C.1. Section C.2 will then briefly discuss the export of linear solvers, which are needed for the IRK methods. Eventually, the newly implemented user-interface `SIMexport` is presented in Section C.3.

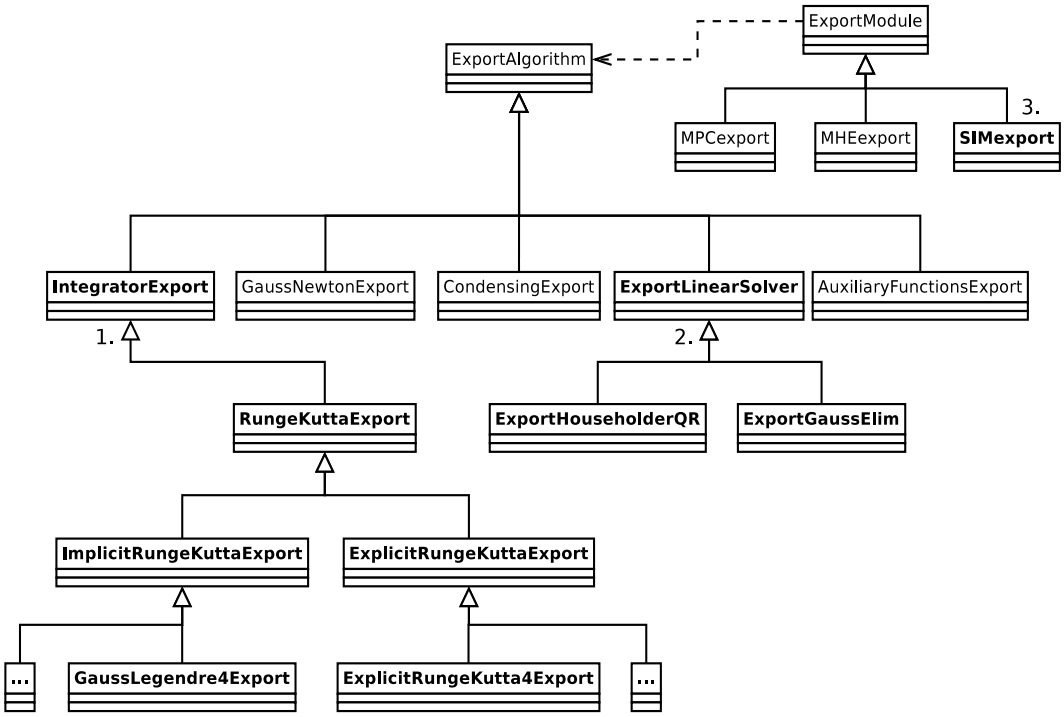



Figure C.1: Inheritance diagram to highlight the most important classes, implemented in the context of this thesis project.

## C.1 IntegratorExport

As can be seen in Figure C.1, an inheritance tree structure has been implemented for the code generation of integration methods. Previously, there was only one class which provided code generation for the ERK method of order 4. To be able to choose from a suite of integrators to be exported, `IntegratorExport` now forms the root of a subtree. The class itself inherits from the class `ExportAlgorithm` which represents any auto generated algorithm. `IntegratorExport` is an abstract class with the following pure virtual functions:

- `setup( )`: initializes the export of a tailored integrator.
- `setDifferentialEquation( Expression& rhs )`: sets the system of equations to be solved by the integrator.
- `getDataDeclarations( ExportStatementBlock& declarations, ExportStruct dataStruct )`: adds all data declarations of the auto generated integrator to the given list of declarations. This pure virtual function is inherited from the class `ExportAlgorithm`.
- `getFunctionDeclarations( ExportStatementBlock& declarations )`: adds all function declarations of the auto generated integrator to the given

list of declarations. This pure virtual function is inherited from the class `ExportAlgorithm`.

- `getCode( ExportStatementBlock& code )`: exports source code of the auto generated integrator. This pure virtual function is also inherited from the class `ExportAlgorithm`.
- `setupOutput( vector<Grid> outputGrids, vector<Expression> rhs )`: sets up the output using the grid and expression for each output function (see Chapter 5).

You can not make an object of the abstract class `IntegratorExport` and a class derived from this class will also be abstract unless all of these pure virtual functions are overridden. `IntegratorExport` also implements other functions which are not pure such as the function `setGrid( Grid& grid, uint numSteps )` which sets the grid for the integrator using the grid of the OCP and the total number of integration steps. Note that the function `setGrid` also supports a non-equidistant control grid if each control interval corresponds to an integer times the fixed integration step size. The class also provides typical getters and a function `hasEquidistantGrid( )` which returns whether the grid is equidistant.

A separate subclass of `IntegratorExport` could then be implemented for every type of integrator that can be exported. For now only RK methods are used in the code generation tool, which is why `RungeKuttaExport` is currently the only subclass of `IntegratorExport`. The major contributions of the `RungeKuttaExport` class are the protected data members representing the Butcher table and the protected function `initializeButcherTableau( )` which initializes this Butcher table. This function is pure virtual and the class does not override the pure functions inherited from `IntegratorExport`, which means that `RungeKuttaExport` is still an abstract class. It even has two abstract subclasses `ExplicitRungeKuttaExport` and `ImplicitRungeKuttaExport` for the code generation of respectively ERK and IRK methods. They both override the functions `setup`, `setDifferentialEquation`, `getDataDeclarations`, `getFunctionDeclarations`, `getCode` and `setupOutput`, but not the function `initializeButcherTableau`. The abstract class for the export of IRK methods also implements support for algebraic equations while the one for the ERK methods currently does not.

For every ERK method that can be exported, a subclass of `ExplicitRungeKuttaExport` provides an implementation of the function `initializeButcherTableau`. It is therefore very easy to add a RK method to the suite of integrators, when its Butcher table is known. Currently, the ERK methods available in the code generation tool are the following:

- `ExplicitEulerExport`: the explicit Euler method of order 1
- `ExplicitRungeKutta2Export`: the ERK method of order 2

- `ExplicitRungeKutta3Export`: the ERK method of order 3 of Heun
- `ExplicitRungeKutta4Export`: the ERK method of order 4

Butcher tables of well-known ERK and IRK methods can be found respectively in [32] and [33]. Similar to the ERK methods, an IRK method can be added by implementing a subclass of `ImplicitRungeKuttaExport`. Note that currently none of the ERK methods supports continuous output, while all of the IRK methods are collocation methods. The IRK methods which are available in the code generation tool are namely:

- `RadauIIA1Export`: Radau IIA method of order 1
- `GaussLegendre2Export`: Gauss method of order 2
- `RadauIIA3Export`: Radau IIA method of order 3
- `GaussLegendre4Export`: Gauss method of order 4
- `RadauIIA5Export`: Radau IIA method of order 5
- `GaussLegendre6Export`: Gauss method of order 6
- `GaussLegendre8Export`: Gauss method of order 8

## C.2 ExportLinearSolver

A variant of the Newton method is used within the IRK methods to solve the nonlinear system. This means that in this case also optimized code for a linear solver needs to be exported. Therefore the abstract class `ExportLinearSolver` has been implemented as a next subclass of `ExportAlgorithm`. It represents all the linear solvers that can be exported and the principle is to generate optimized code for a linear system of a specific dimension. It is mandatory for every solver to have the option to also export code that solves a linear system reusing as many results as possible from a previously solved linear system with the same matrix but a different right-hand side. Another mandatory option is to be able to choose between unrolling all loops in the code of the linear solver and unrolling the least possible. If no decision is made by the user, this will be decided using a simple heuristic. `ExportLinearSolver` is an abstract class with the following pure virtual functions:

- `setup( )`: initializes the export of a tailored linear solver.
- `getDataDeclarations( ExportStatementBlock& declarations, ExportStruct dataStruct )`: adds all data declarations of the auto generated linear solver to the given list of declarations. This pure virtual function is inherited from the class `ExportAlgorithm`.

- `getFunctionDeclarations( ExportStatementBlock& declarations )`: adds all function declarations of the auto generated linear solver to the given list of declarations. This pure virtual function is inherited from the class `ExportAlgorithm`.
- `getCode( ExportStatementBlock& code )`: exports source code of the auto generated linear solver. This pure virtual function is also inherited from the class `ExportAlgorithm`.

For every linear solver which can be exported, a subclass of `ExportLinearSolver` needs to be created which provides an implementation for all of these functions. Currently, only two different linear solvers are available in the code generation tool. The first one is implemented in the class `ExportGaussElim`, makes use of the Gaussian elimination and is preferable here as is explained in Section 2.4.2. This implementation can also export optimized code to solve a linear system, reusing a previously computed LU factorization of the matrix. Just to provide an alternative, also the subclass `ExportHouseholderQR` has been implemented. This linear solver computes the QR factorization of the matrix using Householder triangularization. It can also generate optimized code to solve a linear system, reusing a previously computed QR factorization of the matrix.

## C.3 SIMexport

The goal of the class `SIMexport` is to provide a quick way to check the performance of a specific integrator on a model and to possibly use the auto generated integration code outside of ACADO. It inherits from the class `ExportModule` which represents the export of any set of automatically generated, tailored algorithms. The two major public functions provided by `SIMexport` are the following:

- `exportCode( String& dirName, String& realString, String& intString, int precision )`: exports all the auto generated simulation code into the given directory.
- `exportAndRun( String& dirName, String& initStates, String& controls, String& results, String& ref )`: exports all the auto generated simulation code into the given directory and runs a test to evaluate the performance of the exported integrator. Summarized results of this test are written to the standard output, while the raw data are written to the given files.

An example of the use of this simulation export module can be found in the following source code:

```
#include <acado_code_generation.hpp>

int main() {
    USING_NAMESPACE_ACADO
```

```
//
// DEFINE THE VARIABLES:
//
DifferentialState    xT;      // the trolley position
DifferentialState    vT;      // the trolley velocity
IntermediateState    aT;      // the trolley acceleration
DifferentialState    xL;      // the cable length
DifferentialState    vL;      // the cable velocity
IntermediateState    aL;      // the cable acceleration
DifferentialState    phi;     // the excitation angle
DifferentialState    omega;   // the angular velocity
DifferentialState    uT;      // trolley velocity control
DifferentialState    uL;      // cable velocity control

Control              duT;
Control              duL;

//
// DEFINE THE PARAMETERS:
//
const double         tau1 = 0.012790605943772;
const double         a1   = 0.047418203070092;
const double         tau2 = 0.024695192379264;
const double         a2   = 0.034087337273386;
const double         g = 9.81;
const double         c = 0.0;
const double         m = 1318.0;

//
// DEFINE THE MODEL EQUATIONS:
//
DifferentialEquation    f;
aT = -1.0 / tau1 * vT + a1 / tau1 * uT;
aL = -1.0 / tau2 * vL + a2 / tau2 * uL;

f << dot( xT )    == vT;
f << dot( vT )    == aT;
f << dot( xL )    == vL;
f << dot( vL )    == aL;
f << dot( phi )   == omega;
f << dot( omega ) == 1.0/xL*(-g*sin(phi)-aT*cos(phi)
                     -2*vL*omega-c*omega/(m*xL));
f << dot( uT )    == duT;
f << dot( uL )    == duL;
```

```
    //
    // DEFINE THE OUTPUT MODEL:
    //
    OutputFcn    h;
    h << xT;
    h << xL;

    Vector Meas(1);
    Meas(0) = 10;

    //
    // DEFINE THE WEIGHTING MATRICES:
    //
    Matrix Q  = eye(8);
    Matrix R  = eye(2);
    Matrix P  = eye(8);

    //
    // SET UP THE OPTIMAL CONTROL PROBLEM:
    //
    OCP ocp( 0.0,0.1,5 );
    ocp.minimizeLSQ        ( Q,R );
    ocp.minimizeLSQEndTerm( P    );

    ocp.subjectTo( f );
    ocp.addOutput( h );
    ocp.setupOutput( Meas );

    //
    // SET UP THE SIMULATION EXPORT MODULE:
    //
    SIMexport T(ocp);
    T.set( INTEGRATOR_TYPE, INT_IRK_GL4 );
    T.set( NUM_INTEGRATOR_STEPS, 5 );
    T.set( IMPLICIT_INTEGRATOR_NUM_ITS, 2 );
    T.set( IMPLICIT_INTEGRATOR_NUM_ITS_INIT, 0 );
    T.set( IMPLICIT_INTEGRATOR_NUM_ALG_ITS, 1 );
    T.set( IMPLICIT_INTEGRATOR_NUM_ALG_ITS_INIT, 2 );
    T.setTimingSteps( 10000 );
    T.exportAndRun( "export" );

    return 0;
}
```

After calling the function `exportAndRun`, the following summarized test results are written to the standard output:

```
AVERAGE DURATION OF ONE INTEGRATION STEP:     12.3 µs


STATES:
MAX ERROR AT 0.02 s:    9.9120e-03
MAX ERROR AT 0.04 s:    8.5990e-03
MAX ERROR AT 0.06 s:    3.4105e-03
MAX ERROR AT 0.08 s:    9.6996e-04
MAX ERROR AT 0.10 s:    3.5213e-04

TOTAL MEAN ERROR:    4.6487e-03


OUTPUT FUNCTION 1:
MAX ERROR AT 0.01 s:    1.1523e-02
MAX ERROR AT 0.02 s:    1.5091e-02
MAX ERROR AT 0.03 s:    3.3740e-03
MAX ERROR AT 0.04 s:    3.1681e-02
MAX ERROR AT 0.05 s:    2.0164e-02
MAX ERROR AT 0.06 s:    5.2391e-02
MAX ERROR AT 0.07 s:    3.8540e-02
MAX ERROR AT 0.08 s:    5.6235e-02
MAX ERROR AT 0.09 s:    2.6847e-02
MAX ERROR AT 0.10 s:    1.4112e-02

TOTAL MEAN ERROR:    2.6996e-02
```

# Appendix D

# Extra numerical results

This appendix presents some extra results from numerical experiments described in Chapter 6.

## D.1 Choice of the Standard Parameters

Similar to Figure 6.4, the results of the experiments in Section 6.3 for the Van der Pol oscillator and the kite ODE model are presented in Figure D.1 and D.2.

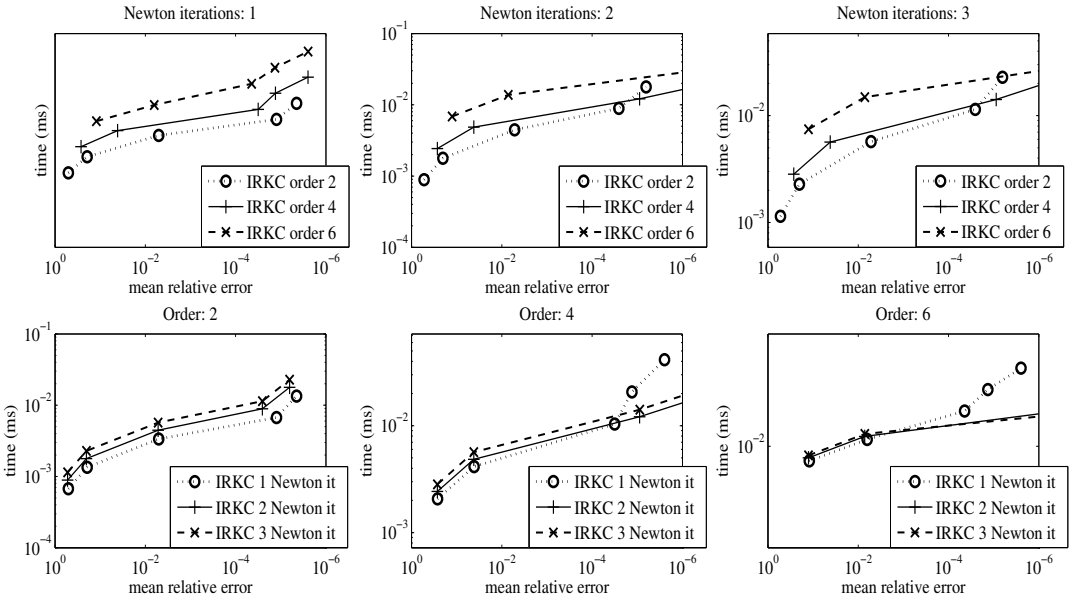



Figure D.1: Van der Pol oscillator: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method.

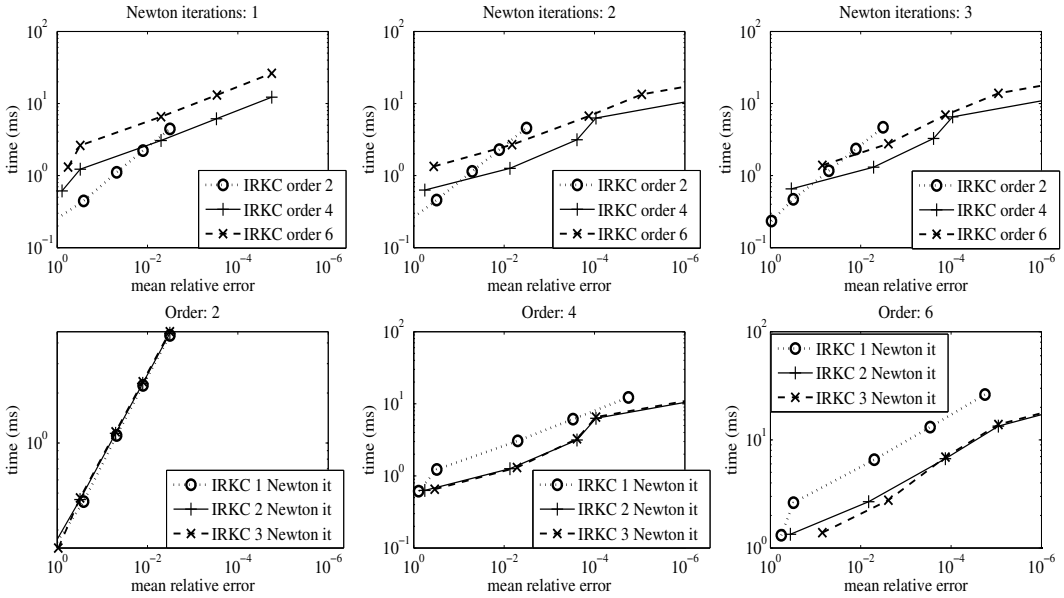



Figure D.2: Elastic kite model: The 6 work-precision diagrams to study the influence of the order of the IRKC method and of the number of Newton iterations on the efficiency of the method.

## D.2 Efficient Derivative Generation

Similar to Figure 6.5, the results of the experiments in Section 6.4 for the Van der Pol oscillator and the crane model are presented in Figure D.3 and D.4.

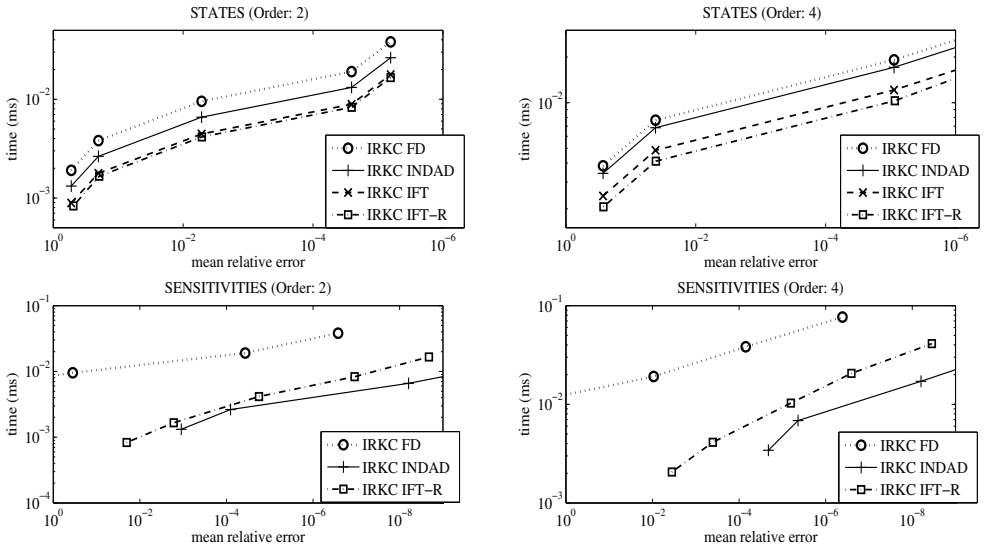



Figure D.3: Van der Pol oscillator: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities.

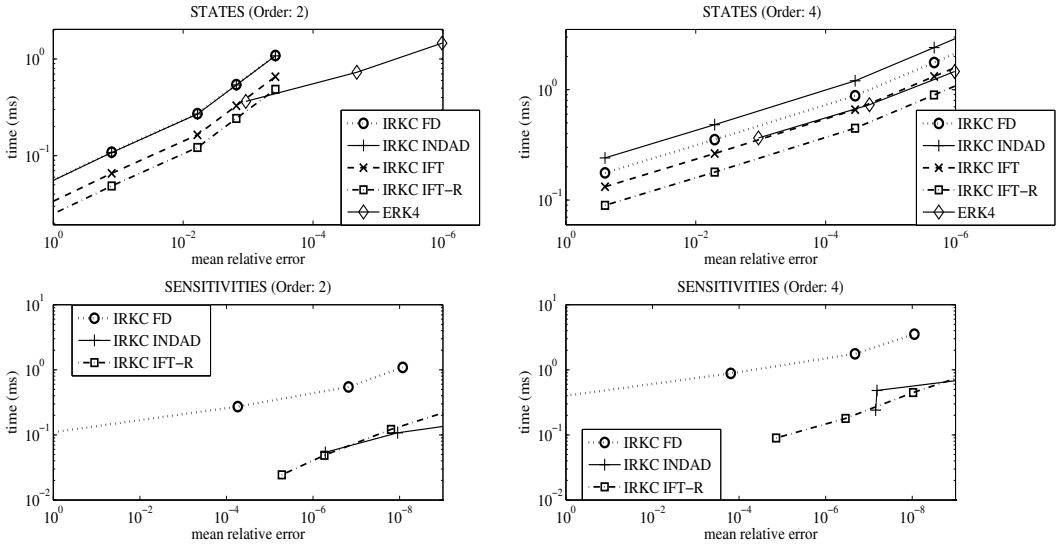



Figure D.4: Crane model: the 4 work-precision diagrams to compare the efficiency of the different methods. The upper and lower diagrams show respectively the accuracy of the computed states and sensitivities.

## D.3 Initialization of Subsequent Steps

Similar to Figure 6.7, the results of the experiments in Section 6.5.2 for the Van der Pol oscillator and the kite ODE model are presented in Figure D.5 and D.6.

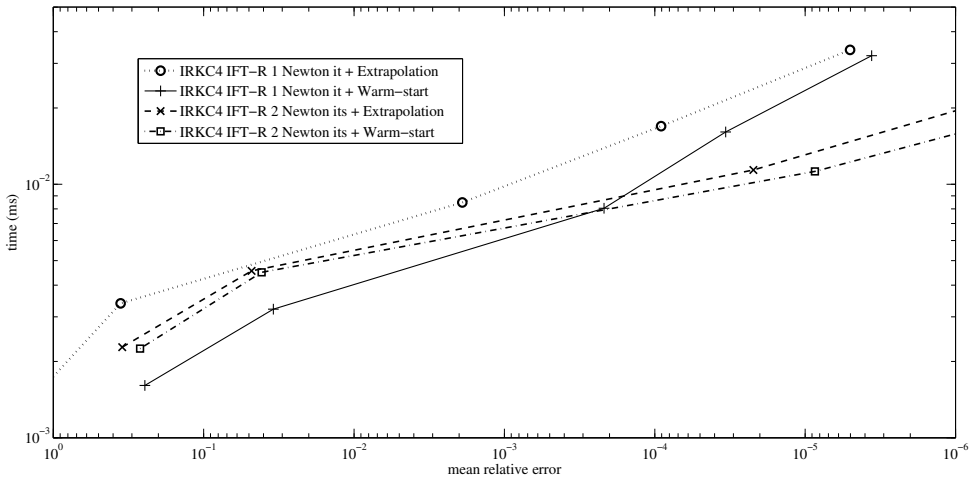



Figure D.5: Van der Pol oscillator: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables.

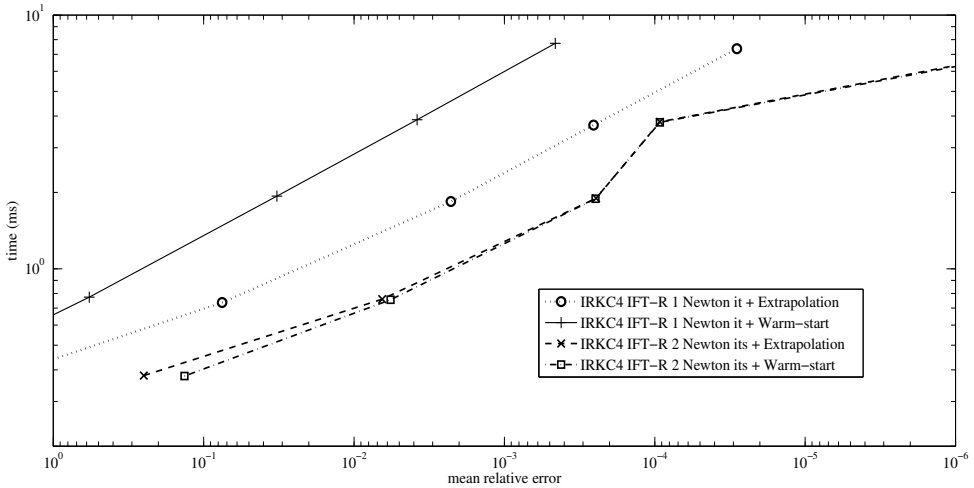



Figure D.6: Elastic kite model: The work-precision diagram for the order 4 IRKC method to study the efficiency impact of using the Extrapolation instead of the Warm-start method to initialize the variables.

## D.4 Solving the Algebraic equations

Similar to Figure 6.8, the results of the experiments in Section 6.6 for the kite DAE model are presented in Figure D.7.

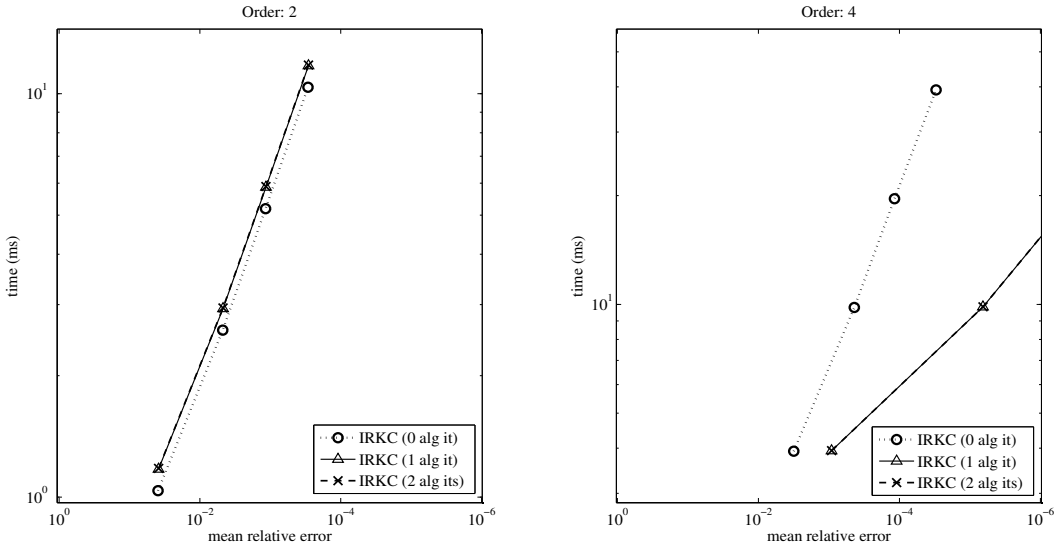



Figure D.7: Kite DAE model: The work-precision diagram for the IRKC IFT-R method to study the impact of the number of Newton iterations for the separate system of algebraic equations.

# Bibliography


[1] J. Albersmeyer and H.G. Bock. Sensitivity Generation in an Adaptive BDF-Method. In H.G. Bock, E. Kostina, X.H. Phu, and R. Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes: Proceedings of the International Conference on High Performance Scientific Computing, March 6-10, 2006, Hanoi, Vietnam*, pages 15–24. Springer, 2008.

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

[3] A. Alessandri, M. Baglietto, G. Battistelli, and V.Zavala. Advances in moving horizon estimation for nonlinear systems. pages 5681– 5688, dec. 2010.

[4] Roger Alexander. Diagonally implicit runge-kutta methods for stiff o.d.e.'s. *SIAM Journal on Numerical Analysis*, 14(6):pp. 1006–1021, 1977.

[5] Mihai Alexe and Adrian Sandu. Forward and adjoint sensitivity analysis with continuous explicit runge–kutta schemes. *Applied Mathematics and Computation*, 208(2):328–346, 2009.

[6] H.G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser, Boston, 1983.

[7] Ian T. Cameron. Solution of differential—algebraic systems using diagonally implicit runge—kutta methods. *IMA Journal of Numerical Analysis*, 3(3):273–289, 1983.

[8] S.L. Campbell and C.W. Gear. The index of general nonlinear DAEs. *Numerische Mathematik*, 72:173–196, 1995.

[9] Y. Cao, S. Li, and L. Petzold. Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software. *Journal of Computational and Applied Mathematics*, 149:171–191, 2002.

[10] Y. Cao, S. Li, L.R. Petzold, and R. Serban. Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and its Numerical Solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.

[11] Nguyen Huu Cong and Le Ngoc Xuan. *Parallel-Iterated RK-type PC Methods with Continuous Output Formulas. International Journal of Computer Mathematics*, 80:8:1025–1035, 2003.

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

[13] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Nonlinear model predictive control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pages 391–417. Springer, 2009.

[14] Moritz Diehl. Real-Time Optimization for Large Scale Nonlinear Processes. PhD thesis, Ruprecht-Karls-Universität, Heidelberg, 2001. URL: http://www.ub.uni-heidelberg.de/archiv/1659/.

[15] Moritz Diehl. *Numerical Optimization Course*, b-kul-h03e3a. Lecture notes, 2009.

[16] Moritz Diehl. *Numerical Optimal Control*. Lecture notes, 2011.

[17] Moritz Diehl, H. Georg Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. *Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. Journal of Process Control*, 12:577–585, 2002.

[18] Moritz Diehl, Peter Kühl, Tom Kraus, Johannes P. Schlöder, and Hans Georg Bock. *A real-time algorithm for moving horizon state and parameter estimation. Computers and Chemical Engineering*, 35:71–83, 2011.

[19] P. Dierckx. *Numerieke methoden voor het oplossen van gewone differentiaalvergelijkingen*. Lecture notes, 2010.

[20] W. H. Enright, K. R. Jackson, S. P. Nørsett, and P. G. Thomsen. Interpolants for runge-kutta formulas. *ACM Trans. Math. Softw.*, 12(3):193–218, September 1986.

[21] Ronald M. Errico. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78:2577–2591, 1997.

[22] David Ariens et al. ACADO for Matlab User's Manual. OPTEC, 2010.

[23] H.J. Ferreau. *Model Predictive Control Algorithms for Applications with Millisecond Timescales*. PhD thesis, K.U. Leuven, 2011.

[24] H.J. Ferreau, B. Houska, K. Geebelen, and M. Diehl. Real-time control of a kite-carousel using an auto-generated nonlinear MPC algorithm. In *Proceedings of the IFAC World Congress*, 2011.

[25] H.J. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl. High-speed moving horizon estimation based on automatic code generation. In *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012), Hawaii*, 2012. (submitted).

[26] R. Findeisen and F. Allgöwer. Nonlinear model predictive control for index–one DAE systems. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, pages 145–162, Basel Boston Berlin, 2000. Birkhäuser.

[27] Jason Frank. Numerical modelling of dynamical systems, chapter 7: *Collocation Methods*. Lecture notes, URL: http://homepages.cwi.nl/~jason/Classes/numwisk/index.html (last checked on July 2, 2012), 2008.

[28] Jason Frank. Numerical modelling of dynamical systems, chapter 8: *Runge-Kutta Methods*. Lecture notes, URL: http://homepages.cwi.nl/~jason/Classes/numwisk/index.html (last checked on July 2, 2012), 2008.

[29] James E. Gentle. *Statistics and Computing:* Numerical Linear Algebra for Applications in Statistics. Springer-Verlag, 1998. ISBN-13: 978-0387985428.

[30] Gene H. Golub and Charles F. Van Loan. Matrix Computations*, Third Edition*. The Johns Hopkins University Press, 1996. ISBN 0-8018-5414-8.

[31] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.

[32] E. Hairer, Syvert P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I:* Nonstiff Problems. Springer, 1986. ISBN 978-3-540-56670-0.

[33] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II:* Stiff and Differential-Algebraic Problems. Springer-Verlag, 1991. ISBN 3-540-53775-9.

[34] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31:363–396, 2005.

[35] 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*, 32(3):298–312, 2011.

[36] Boris Houska and Hans Joachim Ferreau et al. ACADO Toolkit User's Manual. OPTEC, 2011.

[37] C.N. Jones and M. Morari. Polytopic approximation of explicit model predictive controllers. *IEEE Transactions on Automatic Control*, 55(11):2542–2553, 2010.

[38] C. Kirches. A Numerical Method for Nonlinear Robust Optimal Control with Implicit Discontinuities and an Application to Powertrain Oscillations. Diploma thesis, University of Heidelberg, October 2006.

[39] C. Kirches, L. Wirsching, S. Sager, and H.G Bock. Efficient numerics for nonlinear model predictive control. In M. Diehl, Francois F. Glineur, and E. Jarlebring W. Michiels, editors, *Recent Advances in Optimization and its Applications in Engineering*, pages 339–357. Springer, 2010.

[40] Tom Kraus. Real-Time State And Parameter Estimation for NMPC-Based Feedback Control With Application To The Tennessee Eastman Benchmark Process. Master's thesis, University of Heidelberg, 2007.

[41] M.R. Kristensen, J.B. Jørgensen, P.G. Thomsen, and S.B. Jørgensen. An ESDIRK method with sensitivity analysis capabilities. *Computers and Chemical Engineering*, 28:2695–2707, 2004.

[42] M. Kvasnica, I. Rauova, and F. Miroslav. Automatic code generation for real-time implementation of model predictive control. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design, Yokohama, Japan*, 2010.

[43] J. Mattingley and S. Boyd. *CVXGEN webpage*. URL: http://cvxgen.com, last checked on July 2, 2012.

[44] J. Mattingley and S. Boyd. *Convex Optimization in Signal Processing and Communications*, chapter Automatic Code Generation for Real-Time Convex Optimization. Cambridge University Press, 2009.

[45] J. Mattingley, Y. Wang, and S. Boyd. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems Magazine*, 31(3):52–65, 2011.

[46] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: stability and optimality. *Automatica*, 26(6):789–814, 2000.

[47] Richard M. Murray. *Networked Control Systems: Moving Horizon Estimation*, chapter 3 lecture notes. URL: http://www.cds.caltech.edu/~murray/wiki/index.php/NCS:_Moving_Horizon_Estimation, last checked on July 2, 2012.

[48] T. Ohtsuka. A continuation/gmres method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574, 2004.

[49] T. Ohtsuka and A. Kodama. Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers*, 38(7):617–623, 2002.

[50] T. Oohori and A.Ohuchi. An efficient implementation of karmarkar's algorithm for large sparse linear programs. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1384–1392, 1988.

[51] OPTEC. *K.U.Leuven Center of Excellence: Optimization in Engineering*. URL: http://www.kuleuven.be/optec/, last checked on July 2, 2012.

[52] C.V. Rao and J.B. Rawlings. Nonlinear Moving Horizon State Estimation. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, pages 45–69, Basel Boston Berlin, 2000. Birkhäuser.

[53] V.H. Schulz, H.G. Bock, and M.C. Steinbach. Exploiting invariants in the numerical solution of multipoint boundary value problems for DAEs. *SIAM Journal on Scientific Computing*, 19:440–467, 1998.

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

[55] L.L. Simon, Z.K. Nagy, and K. Hungerbuehler. *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, chapter Swelling Constrained Control of an Industrial Batch Reactor Using a Dedicated NMPC Environment: OptCon, pages 531–539. Springer, 2009.

[56] Prof. V. Stejskal, Prof. P. Dehombreux, Prof. A. Eiber, Prof. R. Gupta, and Prof. M. Okrouhlik. Mechmat: *Mechanics with Matlab*. URL: http://www.fsid.cvut.cz/en/u2052/mechmat1.html (last checked on July 2, 2012), 2001.

[57] Lloyd N. Trefethen and David Bau. Numerical Linear Algebra. SIAM, 1997. ISBN 0-89871-361-7.

[58] M. Vukov, W. Van Loock, B. Houska, H.J. Ferreau, J. Swevers, and M. Diehl. Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation. In *The 2012 American Control Conference, Montreal, Canada.*, 2012. (accepted).

[59] Mario Zanon, Moritz Diehl, and Sébastien Gros. *Model Predictive Control of Tethered Planes for Wind Power Generation*. OPTEC Presentation.

# Fiche masterproef

*Student*: Rien Quirynen

*Titel*: Automatic code generation of Implicit Runge-Kutta integrators with continuous output for fast embedded optimization

*Nederlandse titel*: Automatische generatie van code voor Impliciete Runge-Kutta integratoren met continue output voor snelle ingebedde optimalisatie

*UDC*: 51-7

*Korte inhoud*:
Het principe van automatisch geoptimaliseerde code te laten genereren heeft zich recent verspreid binnen de wereld van de ingebedde optimalisatie. Een voorbeeld is de code generation tool in de ACADO Toolkit, die het Real-Time Iteration (RTI) algoritme implementeert. Het simuleren van het systeem en het genereren van de *sensitivities* vormt daarbij een belangrijke computationele stap. Tot nu toe werd hiervoor geoptimaliseerde code voor de expliciete Runge-Kutta (ERK) methode van orde 4 geëxporteerd. Deze masterproef kan worden beschreven door middel van 3 belangrijke bijdragen. Ten eerste wordt er aangetoond dat automatische generatie van impliciete Runge-Kutta (IRK) methoden met de berekening van de *sensitivities* ook efficiënt kan worden geïmplementeerd. Deze methoden zorgen dan voor een betere ondersteuning van stijve stelsels en kunnen uitgebreid worden naar stelsels met ook algebraïsche vergelijkingen. De tweede bijdrage bestaat dan ook uit het efficiënt uitbreiden van de implementatie naar DAE stelsels met index 1.

Collocatie methoden vormen een speciaal geval van IRK methoden die een continue benadering voor de oplossing kunnen bieden en zijn veelbelovend voor MHE met hoog frequente metingen. Ze laten namelijk toe om bij de integratie een stapgrootte te gebruiken die groter is dan het tijdsinterval tussen twee metingen, terwijl toch nog alle meetresultaten zonder verlies gebruikt kunnen worden. De continue output van deze methoden kan nog nuttig zijn voor vele andere toepassingen en de algemene implementatie hiervan vormt het derde aspect van deze masterproef. Het uiteindelijke resultaat van het project naast deze tekst, is de goed gedocumenteerde *open source* implementatie die deel uitmaakt van de ACADO code generation tool.

Thesis voorgedragen tot het behalen van de graad van Master in de ingenieurswetenschappen: wiskundige ingenieurstechnieken

*Promotor*: Prof. Dr. Ir. Moritz Diehl

*Assessoren*: Prof. Dr. Ir. Stefan Vandewalle
Prof. Dr. Ir. Paul Dierckx

*Begeleiders*: Ir. Milan Vukov
Dr. Hans Joachim Ferreau