Auto Generation of Implicit Integrators for Embedded NMPC with Microsecond Sampling Times¹

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 pre-computation, 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:

$$\left\{ \begin{array}{l} k_1 = f(t_n + c_1 h, x_n + h \sum_{j=1}^s a_{1j} k_j), \\ \dots \\ k_s = f(t_n + c_s h, x_n + h \sum_{j=1}^s a_{sj} k_j), \\ x_{n+1} = x_n + h \sum_{i=1}^s b_i k_i, \end{array} \right.\tag{2}$$

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:

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

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:

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

where $k = (k_1, \dots, k_s)$ 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

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

The assumption here is that a fixed number of L Newton iterations is performed and that the initial guess $k^{[0]}$ for the variables $k = (k_1, \dots, k_s)$ 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:

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

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:

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

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

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

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

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 :

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

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

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

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

```

1:  $w_n \leftarrow (x, u)_n$ 
2: if  $n = 0$  then
3:    $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[0]})$ 
4:   factorize  $M$ 
5:    $K^{[0]} \leftarrow K^{[0]} - M^{-1}G(w_n, K^{[0]})$ 
6: end if
7: for  $i = 1 \rightarrow L$  do
8:    $K^{[i]} \leftarrow K^{[i-1]} - M^{-1}G(w_n, K^{[i-1]})$ 
9: end for
10:  $x_{n+1} \leftarrow x_n + h \sum_{i=1}^s b_i k_i$ 
11: predict  $z_{n+1}$  using  $z_n$  and  $Z_i$  values
12: if  $n = 0$  then
13:    $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$ 
14:   factorize  $N$ 
15: end if
16:  $z_{n+1} \leftarrow z_{n+1} - N^{-1}g(t_{n+1}, x_{n+1}, z_{n+1})$ 
17:  $M \leftarrow \frac{\partial G}{\partial K}(w_n, K^{[L]})$  and  $N \leftarrow \frac{\partial g}{\partial z}(t_{n+1}, x_{n+1}, z_{n+1})$ 
18: factorize  $M$  and  $N$ 
19: compute sensitivities using  $M$  and  $N$  in (10)
20: initialize  $K^{[0]}$  for next integration step
21:  $n \leftarrow n + 1$ 

```

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

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

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 μ s	23.3 μ s	12.7 μ s	8.0 μ s

Table 2. Composition of the average computation time per integration step for the 4th order Gauss method on crane model.

method. It computes the sensitivities up to machine precision without the need for an extra LU factorization, except for the first integration step.

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 4th 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.

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

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

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

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

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.

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 (INFOS - 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. Deuffhard 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.