Fast auto generated ACADO integrators and application to MHE with multi-rate measurements

Rien Quirynen, Sébastien Gros and Moritz Diehl

Abstract—Algorithms for real-time, embedded optimization need to run within tight computational times, and preferably on embedded control hardware for which only limited computational power and memory is available. A computationally demanding step of these algorithms is the model simulation with sensitivity generation. This paper presents an implementation of code generation for Implicit Runge-Kutta (IRK) methods with efficient sensitivity generation, which outperforms other solvers for the targeted applications. The focus of this paper will be on the extension of the proposed tool to the integration of index-1 Differential Algebraic Equations (DAE), and continuous output functions, which are crucial for e.g. performing sensor fusion with measurements provided at very high sampling rates. The new tool is provided with a powerful MATLAB interface. It is illustrated in simulation for the trajectory estimation of a mechanical system modeled by complex Differential-Algebraic equations, using sensor information provided at fast, multi-rate sampling frequencies.

Keywords : code generation, IRK methods, sensitivity generation, continuous output, MHE, multi-rate measurements

I. INTRODUCTION

Moving Horizon Estimation (MHE) [1] and Nonlinear Model Predictive Control (NMPC) [2], [3] are popular for their ability to explicitly handle constraints and nonlinear dynamics. For fast systems, the high computational burden of MHE and NMPC is however a major challenge. Both MHE and NMPC involve solving an optimization problem at each sampling time, while respecting the time limitation imposed by the real-time implementation. Recent algorithmic progresses [4], [5], [6] allow considering MHE and NMPC for very fast systems. Among the various available algorithms, the Real-Time Iteration (RTI) scheme [7] has been proposed as a highly competitive approach to fast MHE and NMPC.

The dominating computational component of these algorithms is often the simulation of the model equations, with sensitivity generation. For real-time optimization, in order to guarantee a deterministic runtime, an integration method with fixed order and step size is preferable. In a multiple-shooting framework, integration takes place over relatively small time intervals. In such a context, single-step methods such as Runge-Kutta (RK) schemes are especially attractive, because they do not need a start-up procedure. Moreover, the Implicit RK (IRK) methods allow for a natural extension to Differential Algebraic Equations (DAE). An efficient implementation in combination with sensitivity generation has already been presented in [8].

Collocation methods form a specific group of IRK methods with the capability of providing outputs between grid points, at the cost of a single polynomial evaluation. In the context of MHE, continuous output integrators allow for using a larger integration step size than the actual frequency of the measurements, such that high frequency sensor information can be included in the estimation scheme at a very low computational cost.

In addition to using efficient algorithms, code generation allows for a significant improvement of the computational time by removing from the code the unnecessary computations, by optimizing memory accesses, cache usage, and by efficiently exploiting the problem dimensions and sparsity patterns. Code generation is e.g. used in the ACADO toolkit, where highly efficient, self-contained C-code can be automatically exported [9]. The methods presented in this paper are provided within the ACADO code generation tool and are also available from MATLAB.

Contribution: This paper presents an auto generated IRK method for the integration of index-1 DAE systems with efficient sensitivity generation, and the ability to compute outputs on an arbitrary time grid at a very low additional computational cost, hence allowing for considering new applications such as e.g. MHE with multi-rate, high frequency sensor information.

The paper is organized as follows. Section II describes the implementation of auto generated IRK methods for index-1 DAE systems, including a discussion on sensitivity generation and continuous output. A user friendly MATLAB interface is then presented in Section III with a tutorial use case. Section IV then illustrates the performance of the integrators on an estimation problem with a focus on the continuous output feature and the corresponding sensitivity generation.
II. AUTO GENERATED IRK METHODS

This section presents code generation for IRK methods with efficient sensitivity generation. The formulation of these methods for ODE and DAE systems of index 1 is treated in Subsection II-A. Some implementation details are mentioned in Subsection II-B and the efficient computation of sensitivities is discussed in Subsection II-C. Subsection II-D highlights an important extra feature of these auto generated integrators, namely the continuous output.

A. Formulation

The assumption in this paper is that the following Initial Value Problem (IVP) needs to be solved over a certain time interval:

\[ 0 = f(t, \dot{x}(t), x(t), z(t), p(t)), \]
\[ x(0) = x_0, \]

with \( x(t) \) a vector of \( N_x \) differential states and \( \dot{x}(t) \) the corresponding time derivatives, \( z(t) \) a vector of \( N_z \) algebraic states, \( p(t) \) a vector of parameters and \( f \) defines a system of \( N_x + N_z \) equations. The presented integrators are therefore able to handle all models ranging from explicit ODE to implicit DAE systems. Note that this DAE formulation is more general than the one used in [8]. The only assumption here is that the DAE system is of index 1. In that case, the algebraic states \( z \) and the differential state derivatives \( \dot{x} \) are uniquely defined by the parameters and differential states, i.e. the matrix \( \frac{\partial f}{\partial (z, x)} \) is invertible [10].

An \( s \)-stage RK method is defined by the coefficients \( c_i \), the weights \( b_i \) and the internal coefficients \( a_{ij}, \forall i, j = 1, \ldots, s \) which are often presented using a Butcher table. The method is explicit if the matrix \( A \) is strictly lower triangular, otherwise it is implicit. Latter methods involve more computational effort, but they generally also provide a higher order of accuracy and better stability properties. The focus of this paper will be on A-stable IRK methods such as the Gauss or Radau methods [11].

An \( s \)-stage RK method can be applied to the system in (1) for the integration from time point \( t_n \) until \( t_{n+1} = t_n + h \):

\[ 0 = f(t_n + c_i h, k_i, x_n + h \sum_{j=1}^{s} a_{ij} k_j, Z_i, p), \quad i = 1, \ldots, s, \]
\[ x_{n+1} = x_n + h \sum_{i=1}^{s} b_i k_i, \]

with \( Z_i \) the stage values of the algebraic states \( z \) and \( k_i \) the stage values for \( \dot{x} \). This nonlinear system of \( (N_x + N_z) \times s \) equations can then be solved using a variant of the Newton method. Different from the approach in [8], consistency of the state values is here only assured at the \( s \) collocation nodes. One reason for this is the associated computational cost in case of the general system in (1). Another reason is that stiffly accurate IRK methods such as the Radau IIA methods already include the endpoint as one of these nodes.

B. Implementation

Solving the nonlinear system in (2) will be done using a variant of the Newton method which always performs a fixed amount of \( L \) iterations. The presented integrators also use a fixed step size and a fixed order, because of the need for a deterministic computation time when targeting real-time applications. Given a good initialization of the variables \( K = (k_1, \ldots, k_s, Z_1, \ldots, Z_s) \), a few iterations are sufficient in practice [8]. Since successive values of the Jacobian will not differ much in this case, only one evaluation is done per integration step.

Let us write the nonlinear system as \( G(x_n, K) = 0 \). The Newton iterations can then be summarized as

\[ M = \frac{\partial G}{\partial K}(x_n, K^{[0]}) \]
\[ K^{[i]} = K^{[i-1]} - M^{-1}G(x_n, K^{[i-1]}), \quad i = 1, \ldots, L \]  

The auto generated code therefore consists of two major components, namely the evaluation of the Jacobian and the solution of a linear system. The Jacobian \( \frac{\partial G}{\partial K}(x_n, K) \) is a \( s(N_x + N_z) \times s(N_x + N_z) \) matrix with the following structure:

\[
\begin{pmatrix}
\frac{\partial f_1}{\partial x} + ha_{11} \frac{\partial f_1}{\partial z} & \cdots & ha_{1s} \frac{\partial f_1}{\partial z} & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
ha_{s1} \frac{\partial f_s}{\partial x} & \cdots & \frac{\partial f_s}{\partial x} + ha_{ss} \frac{\partial f_s}{\partial z} & 0 & \cdots & 0
\end{pmatrix}
\]

where \( f_i = f(t_n + c_i h, k_i, x_n + h \sum_{j=1}^{s} a_{ij} k_j, Z_i, p) \). Evaluating the derivatives of the function \( f \) is done using Algorithmic Differentiation (AD) [12]. The linear systems are solved using one LU factorization of the matrix \( M \) for the different right-hand sides. It is useful to note the sparsity structure in (4), which could be exploited.

C. Sensitivity generation

In addition to the state values at the next time point, the generated integrators can provide accurate sensitivities in an efficient way. This is crucial in the context of dynamic optimization. Let us assume that \( w_n = (x_n, p) \) denotes all the independent variables with respect to which the first order derivatives are needed. The goal is then to efficiently compute the matrix \( \partial (x_{n+1}, z_{n+1})/\partial w_n \). Since the continuous output is an important feature of the presented methods, only forward techniques will be considered. A thorough discussion on sensitivity generation for IRK methods can be found in [13].

Let us summarize by saying that the use of the Variational Differential Algebraic Equations (VDAE) is not suited for an implicit integration method. Using finite differences can lead to relatively inaccurate results. And the principle of Internal Numerical Differentiation (IND) from [14] results in an iterative scheme, which is costly when many directional derivatives are needed. According to the discussion in [8], a better approach is therefore to abuse the fact that an approximation of the Jacobian \( \partial G/\partial K \) is needed for the Newton iterations in (3) anyway. This means that evaluating this Jacobian to compute sensitivities makes sense if it can be reused in the Newton iterations of the next integration step.
Algorithm 1 One step of the IFT-R implementation

Input: $w_n$, initial $K[0]$, LU factorization of $M$
Output: $(x,z)_{n+1}$ and the possibility to export them as standalone components which can then be used outside of the ACADO environment. The user’s interpolating polynomial for the algebraic states is therefore undesirable. The states at time $t^n$ are obtained by a direct approach. In the IFT-R implementation, the Jacobian is reused in the next integration resulting in only one factorization per step. The latter approach is described by Algorithm 1.

D. Continuous output

Some promising possibilities of auto generated IRK methods with continuous output have already been mentioned in the introduction. Let us focus on the family of collocation methods, which naturally provides a continuous extension. For a collocation method, the coefficients $c_i$ need to be distinct and they completely determine the coefficients $b_i$ and $a_{ij}$. This way, a polynomial can be constructed that passes through $(t_n, x_n)$ and that agrees with the model equations in (1) at the $s$ different nodes. Evaluating this polynomial at some time point $t_n + ch$ results in

$$x(t_n + ch) \approx x_n + h \sum_{i=1}^{s} k_j \int_0^1 l_j(\tau) d\tau,$$

where $l_j(t) = \prod_{j \neq i}^{s} \frac{t - c_i}{c_j - c_i}$ are the Lagrange interpolating polynomials. Note that for $c$ equal to 1, this corresponds to the computation of $x_{n+1}$ in (2). In the case of an $s$-stage IRK method of order $p$, the order of this continuous output is $p^* = \min(p, s + 1)$ [15].

It is possible to do something equivalent for the algebraic states. However, this would mean that consistent values of the states at time $t_n$ are needed which is not evident. In the case of a discontinuous jump in the value of a parameter or control input, the differential states will vary continuously while the algebraic states could also exhibit such a jump. Extra computation time spent to achieve consistency at the beginning of the integration step is undesirable. The interpolating polynomial for the algebraic states is therefore one order less and similar to the one for the differential state derivatives:

$$z(t_n + ch) \approx \sum_{i=1}^{s} l_i(c)Z_i,$$

The conclusion from this should be that the differential as well as the algebraic states can efficiently be computed on a grid finer than the integration grid. The auto generated integrators exploit this by allowing the user to also define one or more output functions like

$$y = \psi(t, x(t), z(t), p(t)),$$

with a corresponding grid on which they should be evaluated. Using the computed derivatives of the variables $K$ from (5), also sensitivities of these extra outputs can be obtained. This is important when one would like to use these outputs in an optimization problem, as will be illustrated in Section IV.

III. THE MATLAB INTERFACE

The presented integrators are available in the code generation tool of the ACADO Toolkit [16], [17]. It is implemented in C++ and the exported C-code is self-contained and efficient, since it is tailored to the specific problem formulation. The motivation behind a MATLAB interface to this code generation tool for integrators is discussed in Subsection III-A and a tutorial use case is presented in Subsection III-B.

A. The motivation

The goal of the MATLAB interface to ACADO is first of all to lower the threshold for potential users of advanced dynamic optimization algorithms, who are more familiar with MATLAB than with C++. It is also interesting because of all the other features and algorithms that are available in the MATLAB environment, resulting in easy interfacing and many prototyping possibilities. A limited interface already existed and is described in the ACADO for MATLAB User’s Manual [18].

The motivation for a MATLAB interface is stronger in case of code generation, since high-level access to the efficient, auto generated C-code could be even more valuable [19]. Interfaces to important functions in the C-code can be automatically generated in the form of MEX-files. This allows the user to formulate the problem as well as to run simulations from MATLAB, while keeping the performance of the optimized C-code. The key result is that the exact same code, that has been tested in simulation, can later be used on the embedded hardware.

B. A tutorial use case

The considered test problem is a small nonlinear ODE model for an overhead crane, similar to the one used in [17]. The system has 8 differential states and 2 control inputs. It is possible to export the presented integrators within NMPC, as illustrated in [8]. Let us however focus on the possibility to export them as standalone components which can then be used outside of the ACADO environment. The user’s
Fig. 1. The resulting figure from the tutorial use case, showing the maximum relative error of the simulated states over 5s.

The goal here would be to get an idea about the performance of different methods and the corresponding step size to be used for this model.

a) Formulate the model: After defining the states and control inputs, the expressions of the ODE system can be constructed as follows:

```matlab
DifferentialState xT vT xL vL phi omega uT uL;
Control duT duL;

tau1 = 0.0128; a1 = 0.0474;
tau2 = 0.0247; a2 = 0.0341;
h = 0.01; g = 9.81; m = 1318;
aT = -1.0/tau1*vT + a1/tau1*uT;
aL = -1.0/tau2*vL + a2/tau2*uL;
f = [ vT; aT; vL; aL; omega; ... -1.0/xL^2*(g*sin(phi)+aT*cos(phi)+ ... 2*vL*omega); duT; duL ];
sim = acado.SIMexport( h );
sim.setMeasurements(100);
```

In accordance with the discussion of Subsection II-D, it is also possible to define one or more output functions with their corresponding amount of measurements per interval. An example of this is the following:

```matlab
sim.addOutput([xT; vT; xL; vL]);
sim.setMeasurements(100);
```

b) Specify the integrator: The step size should be chosen as well as the specific integration method to be exported. Examples of available IRK schemes are the Radau IIA methods of order 1, 3 and 5 and the Gauss methods of order 2, 4, 6 and 8. Also explicit RK schemes are available with orders ranging from 1 until 4. Let us export the Gauss method of order 4 with only one step of size $h = 0.01$:

```matlab
sim.exportCode;
sim.set( 'NUM_INTEGRATOR_STEPS', 1 );
sim.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL4' );
sim.setMeasurements(100);;
sim.set( 'NUM_INTEGRATOR_STEPS', 1 );
```

c) Export and use the code: It is now possible to let the tool export and compile the code, together with a MEX-

function to call the integrator and one to call only the right-hand side of the model. The latter can be used with one of the standard MATLAB integrators such as ode45 or ode15s. This allows one to easily use the solver, while keeping the performance of the optimized C-code. It is for example possible to do some simulations and compare the results with those of a MATLAB integrator:

```matlab
sim.exportCode;
sim.set( 'NUM_INTEGRATOR_STEPS', 1 );
x = rand(8,1); u = zeros(2,1);
xs = x';
N = 500;
for i = 1:N
    [states out] = integrate(xs(:,i),u);
x = states.out;
end

options=odeset('RelTol',1e-12,'AbsTol',1e-12);
[tout exact] = ode15s(@(t,y) rhs(t, y, u), ...
    [0:h:N*h],x,options);
err = max(abs(xs-exact),[],2);
semilogy([h:h:N*h], err(2:end), ':kx');
```

The eventual results of this tutorial when running the MATLAB code are shown in Figure 1.

IV. APPLICATION: MHE WITH MULTI-RATE MEASUREMENTS

This section presents an example using the integrators for MHE with multi-rate measurements. Figure 2 depicts the set-up that is considered. The system consists of a cube connected to a rod with one of its corners. The model is developed using multi-body modeling and a rotation-less formulation, where the position of the center of mass of the cube $p = [x, y, z]$ and the orientation of the cube given by the rotation matrix $R \in \mathbb{R}^{3 \times 3}$ are the generalized coordinates. In this framework, the kinetic and potential energy of the cube are given by:

$$
T = \frac{1}{2} m \dot{p}^T \dot{p} + \frac{1}{2} \omega^T J \omega, \quad V = mgz
$$

where $m$ is the mass of the cube, $J \in \mathbb{R}^{3}$ is the inertia tensor of the cube and $\omega \in \mathbb{R}^3$ is the angular velocity of the cube, both in its reference frame. The motion of the cube is constrained by the rod to evolve on the manifold:

$$
c(p, R) = \frac{1}{2} (p^T p + L^2 - L^2) = 0, \quad p_c = p + RC
$$

where $C = [1 \ 1 \ 1]^T$ is the position of the attached corner in the cube reference frame, $p_c \in \mathbb{R}^3$ is the position of the attached corner in the fixed reference frame, and $L = 1$ is the length of the rod.

The Lagrangian of the cube then reads:

$$
\mathcal{L} = T - V - \nu c(p, R) - \text{tr}(Z (R^T R - I))
$$

where $\nu \in \mathbb{R}$ is the Lagrange multiplier associated to the constraint $c$, and $Z \in \mathbb{S}^{3}$ is the symmetric matrix of Lagrange multipliers associated to the orthonormality of the rotation
matrix $R \in SO(3)$. Defining the linear operator $P : \mathbb{R}^{3 \times 3} \times \mathbb{R}^{3 \times 3} \to \mathbb{R}^{3}$, $P(A, R) = U(R^T A)$ where
\[
U \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} a_{32} - a_{23} \\ a_{13} - a_{31} \\ a_{21} - a_{12} \end{pmatrix},
\]
it can be shown [20] that the dynamics of the cube reduce to the following index-1 DAE system:
\[
\dot{R} = R \omega
\]
\[
\begin{bmatrix} J & 0 & \nabla_p c \\ 0 & J & 2P(\nabla_R c)^T \\ \nabla_p c^T & 2P(\nabla_R c)^T & 0 \end{bmatrix} \begin{bmatrix} \dot{\omega} \\ \dot{\omega} \\ \dot{c} \end{bmatrix} = \begin{bmatrix} -\nabla_p V \\ -\omega \times J \omega \\ -\tau (\nabla_R c^T R \omega) + \nabla_p c^T \dot{p} \end{bmatrix},
\]
with the consistency conditions:
\[
c = 0
\]
\[
\dot{c} = (\tau (\nabla_R c^T R \omega) + \nabla_p c^T \dot{p})_{t=t_0} = 0
\]
\[
(R^T R) = I
\]
that must be enforced at any time $t_0$ of the trajectory. In the following, (9) will be lumped into $H = \begin{bmatrix} c & \dot{c} & R^T R - I \end{bmatrix}$.

The goal here is to accurately estimate the position and orientation of the moving cube, using two different types of measurements [21]. Absolute location information is available at a low frequency of 10Hz using the Global Positioning System (GPS). In addition, measurements of the angular velocity and linear acceleration are provided by an Inertial Measurement Unit (IMU) at a high frequency of 800Hz. This estimation problem can be handled by MHE, using a moving horizon formulation. An optimal estimate $x_N$ for the current states of the system is then obtained by solving this dynamic optimization problem at each sampling time:
\[
\text{minimize} \sum_{k=0}^{N-1} \frac{1}{2} \| R_k(x_k) \|_2^2
\]
subject to
\[
x_{k+1} = G_k(x_k), \quad \forall k = 0, \ldots, N-1,
\]
\[
0 = H(x_N),
\]
where $R_k$ contains the difference between the real measurements and the outputs from the integrator and $G_k$ corresponds to the integration from one GPS sample to the next. There are also 8 extra equality constraints that need to be satisfied. By imposing them at the end point $H(x_N) = 0$, the dynamics of the system will make sure they are satisfied over the whole horizon.

The result is an Optimal Control Problem (OCP) with a least squares objective and only equality constraints. Let us assume that the number of estimation intervals $N = 10$ and the horizon is 1s. This OCP can be solved at each time point using multiple shooting in combination with the Constrained Gauss-Newton (CGN) method from [22]. Using the new integrators and corresponding interface, this algorithm can be implemented in an easy but still efficient way in MATLAB. The simulated values corresponding to the high frequency IMU measurements can be obtained using the continuous output feature as previously presented in [23]. This allows us to use an integration step size that is significantly larger than the time between two IMU measurements without any loss of information. In addition to these values, their sensitivities are also needed for the linearization done by the CGN method.

Figure 3 shows some results of a simulation over 2s. The simulated values of the linear velocity of the cube are presented as well as the estimates from the MHE algorithm. To the GPS and IMU measurements that are used in this simulation, Gaussian noise has been added with a standard deviation equal to 5% of that of the signal. It is clear from the figure that the MHE algorithm is still estimating the states very well. It is of course possible to implement the same algorithm using a standard DAE solver from MATLAB such as ode15s. The sensitivities of the outputs can then be obtained approximately using finite differences. A comparison between both implementations is given in Table I. The numerical experiments are run on an ordinary computer (Intel P8600 3MB cache, 2.40 GHz, 64-bit Ubuntu 12.04 LTS and MATLAB R2011a 64 bit) and the time measurements are done using the tic/toc functions in MATLAB.

A Radau IIA method of order 3 and fixed step size 0.02s is exported using ACADO, while MATLAB’s ode15s implements a variable-order, variable-step method. The ACADO integrators can provide the sensitivities in an accurate and efficient way while ode15s needs to be called 19 times.
Table I
One CGN iteration using respectively an auto generated ACADO solver and MATLAB’s DAE solver ode15s.

<table>
<thead>
<tr>
<th></th>
<th>ACADO</th>
<th>ode15s</th>
</tr>
</thead>
<tbody>
<tr>
<td>integrator calls</td>
<td>10</td>
<td>190</td>
</tr>
<tr>
<td>integration + sensitivities</td>
<td>0.02 s</td>
<td>10.65 s</td>
</tr>
<tr>
<td>total computation time</td>
<td>0.04 s</td>
<td>10.69 s</td>
</tr>
</tbody>
</table>

(18 states in the system) to obtain relatively inaccurate sensitivities. The accuracy of the sensitivities will strongly influence the convergence behavior of the algorithm, i.e. the amount of iterations. Table I therefore compares the cost of one iteration for the two different implementations. Keep in mind that 10 estimation intervals are being used.

While [8] showed that the ACADO auto generated methods can be approximately 100 times faster than the solvers from SUNDIALS, a speedup of factor 50 is observed here with respect to ode15s. The important conclusion here should not be that the auto generated methods are faster than a standard MATLAB integrator, since that is what would be expected. The complete message is that they are much faster, equally easy available in MATLAB and more powerful because of the sensitivity generation and the continuous output feature. Most importantly, the same auto generated integrator can eventually be used in a real-time environment.

V. CONCLUSIONS & FURTHER DEVELOPMENTS

This paper has proposed code generation for IRK methods with efficient computation of sensitivities and a powerful continuous output feature. A compact discussion on the formulation and implementation of IRK methods for index-1 DAE systems has been given. The implementation of sensitivity generation and of the continuous output has been discussed. Also a MATLAB interface to this code generation tool for integrators has been presented and a nontrivial multi-rate MHE application has been used to show its power and usefulness. Numerical experiments confirmed that auto generated IRK methods deliver a much better performance than other solvers for the targeted applications. It is eventually important to note that the ACADO integrators are available as open source code [24].

Future work will expand the proposed methods by improving the support for larger problems, exploiting the sparsity present in the linear systems and by implementing code generation for other integration methods. Also the MATLAB interface will be improved and maintained.

REFERENCES