OPTIMAL CONTROL APPLICATIONS AND METHODS
Optim. Control Appl. Meth. 2015; **36**:685–704
Published online 26 November 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/oca.2152

Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators

R. Quirynen[1,*,†], M. Vukov[1], M. Zanon[1,2] and M. Diehl[1,2]

[1]*KU Leuven, ESAT-STADIUS, B-3001 Leuven, Belgium*
[2]*University of Freiburg, IMTEK, 79110 Freiburg, Germany*

SUMMARY

Nonlinear model predictive control (NMPC) allows one to explicitly treat nonlinear dynamics and constraints. To apply NMPC in real time on embedded hardware, online algorithms as well as efficient code implementations are crucial. A tutorial-style approach is adopted in this article to present such algorithmic ideas and to show how they can efficiently be implemented based on the ACADO Toolkit from MATLAB (MathWorks, Natick, MA, USA). Using its code generation tool, one can export tailored Runge–Kutta methods—explicit and implicit ones—with efficient propagation of their sensitivities. The article summarizes recent research results on autogenerated integrators for NMPC and shows how they allow to formulate and solve practically relevant problems in only a few tens of microseconds. Several common NMPC formulations can be treated by these methods, including those with stiff ordinary differential equations, fully implicit differential algebraic equations, linear input and output models, and continuous output independent of the integration grid. One of the new algorithmic contributions is an efficient implementation of infinite horizon closed-loop costing. As a guiding example, a full swing-up of an inverted pendulum is considered. Copyright © 2014 John Wiley & Sons, Ltd.

Received 23 October 2013; Revised 17 October 2014; Accepted 3 November 2014

KEY WORDS: NMPC; embedded optimization; Runge–Kutta methods; code generation; sensitivity analysis

1. INTRODUCTION

In model predictive control (MPC), at each sampling instant, one solves an optimal control problem (OCP) using the current system state as initial value. The increasing popularity of nonlinear MPC (NMPC) for real-time control is due to its ability to directly handle nonlinear dynamics and constraints. Let us consider the following continuous time OCP formulation:

$$\begin{aligned}
\underset{x(\cdot),\,u(\cdot)}{\text{minimize}} \quad & \int_{t_0}^{t_0+T} \|F(t,x(t),u(t))\|_2^2 \, \mathrm{d}t + \|F_N(x(t_0+T))\|_2^2 && \text{(1a)}\\
\text{subject to} \quad x(t_0) &= \bar{x}_0, && \text{(1b)}\\
\dot{x}(t) &= f(t,x(t),u(t)), \qquad \forall t \in [t_0, t_0+T], && \text{(1c)}\\
0 &\geqslant h(x(t),u(t)), \qquad \forall t \in [t_0, t_0+T], && \text{(1d)}\\
0 &\geqslant r(x(t_0+T)), && \text{(1e)}
\end{aligned}$$

where t_0 is the current time instant, $x(t) \in \mathbb{R}^{n_x}$ denotes the differential states, and $u(t) \in \mathbb{R}^{n_u}$ are the control inputs at time t. The NMPC objective is defined by (1a), while (1d) and (1e) denote respectively the path and terminal constraints. The nonlinear dynamics in (1c) are described by an

*Correspondence to: R. Quirynen, KU Leuven, ESAT-STADIUS, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium.
†E-mail: rien.quirynen@esat.kuleuven.be

explicit system of ordinary differential equations (ODEs), but this will be further generalized to implicit systems of differential algebraic equations (DAEs). The optimization problem depends on the parameter $\bar{x}_0 \in \mathbb{R}^{n_x}$ through the initial value constraint of (1b) and can also be time dependent. Hence, the control trajectory obtained by solving problem (1) provides a feedback strategy $u^*(t_0, \bar{x}_0)$, which depends on the current state and time.

The OCP is in practice often solved by a direct approach where one first discretizes the problem to obtain a structured nonlinear program (NLP), which is generally nonconvex. A Newton-type algorithm is able to find a locally optimal solution by solving the Karush–Kuhn–Tucker conditions. Two popular Newton-type techniques are interior point (IP) methods and sequential quadratic programming (SQP) [1]. IP methods for nonconvex problems treat the inequality constraints therein by the use of a smoothening technique [2]. SQP instead consists in sequentially approximating the NLP by convex quadratic program (QP) subproblems.

Recent algorithmic progress [3, 4] allowed to reduce computational delays between receiving the new state estimate and applying the next control input to the process [3]. This made it possible to apply NMPC also to fast dynamic systems with sampling times in the millisecond or even microsecond range. The real-time iteration (RTI) scheme [5] is an SQP-type online algorithm. The resulting sequence of sparse QPs can either be solved directly using a structure exploiting convex solver such as FORCES [6] or qpDUNES [7] or by reducing the size of the QP subproblems with a condensing technique [8, 9] and using a dense linear algebra solver such as qpOASES [10]. A discussion on these algorithmic aspects can be found in [11]. It is important to note that the presented techniques are also applicable to moving horizon estimation, which delivers estimates of process states or parameters in real time [12].

Embedded integrators with efficient sensitivity propagation are a crucial part of the algorithmic tools needed to implement NMPC in real time. The online linearization of constraints imposing the model equations is typically a computationally expensive step in the RTI scheme [13]. Targeting real-time applications, a deterministic run time for the used integration methods is also important. The approach throughout this article will be that parameters such as the step size and order of the method are therefore determined offline and kept fixed. The use of tailored explicit Runge–Kutta (ERK) methods for real-time optimal control has been presented in [13, 14]. This idea was extended in the more recent work on online implicit Runge–Kutta (IRK) methods [15]. Tailored techniques for the efficient computation of the sensitivity information have been discussed in [16]. The application of these embedded, implicit solvers to systems of DAEs and the motivation for continuous output were presented in [17]. A three-stage model formulation is proposed in [18], as a way to exploit common linear subsystems in a dynamic model.

In addition to using tailored algorithms, highly efficient code implementations are necessary to meet the tight timing constraints on embedded control hardware. In this context, the technique of automatic code generation has experienced an increasing popularity [19]. One can perform offline optimizations of the code to be generated, for example, by removing unnecessary computations, by optimizing memory access and cache usage, and by exploiting the problem's specific structure. Convex optimization solvers such as CVXGEN [20] and FORCES [6] are both generating customized IP solvers. In order to solve nonlinear OCP problems, the ACADO code generation tool [13] exports highly efficient C code, implementing a tailored RTI scheme. It is part of the open-source ACADO Toolkit and interfaced to various convex solvers [7, 11]. ACADO code generation has already been used for real-time optimal control, showcasing millisecond or even microsecond execution times both in simulation [21, 22] and in real-world experiments [14, 23–25]. This article will show how to efficiently implement NMPC in a rather intuitive way, by using ACADO from MATLAB (MathWorks, Natick, MA, USA). The focus will mostly be on the code-generated integrators and their importance in an efficient treatment of various NMPC formulations. Novel SQP-type algorithms can also be prototyped easily, by using the stand-alone integrator code in combination with a convex solver.

This paper is organized as follows. Section 2 briefly presents direct multiple shooting and the Gauss–Newton method in the RTI scheme. Section 3 introduces the guiding example, targeting the swing-up of an inverted pendulum. Code generation of explicit and implicit integration methods is respectively discussed and illustrated in Sections 4 and 5. The exploitation and use of a three-stage model structure for NMPC are the topic of discussion in Section 6. Continuous output is then presented by Section 7 and used to efficiently approximate infinite horizon closed-loop costing. Section 8 finally concludes the paper and outlines further developments. All numerical simulations are performed using the ACADO code generation tool on a computer equipped with Intel i7-3720QM processor (Intel, Santa Clara, CA, USA), running a 64-bit version of Ubuntu 12.04 (Canonical Ltd, London, UK).

2. NUMERICAL METHODS FOR NONLINEAR MODEL PREDICTIVE CONTROL

An overview of the problem transformations in our numerical treatment of NMPC is shown in Figure 1. Subsection 2.1 will present direct multiple shooting as a reliable way of reformulating the time continuous OCP as an approximate but tractable NLP. SQP using the popular Gauss–Newton Hessian approximation is described briefly in Subsection 2.2. The principle of the RTI scheme is summarized in Subsection 2.3, although more details can be found in the references therein. The open-source ACADO code generation tool will be presented in Subsection 2.4.

2.1. Multiple shooting discretization

Only direct optimal control methods are treated in this paper, where one first discretizes the OCP in (1) and then solves the resulting optimization problem. Within this family of methods, one can distinguish between sequential and simultaneous approaches. The latter comprises both collocation and *multiple shooting* [26], which results in the following NLP:

$$
\begin{aligned}
\underset{X,U}{\text{minimize}} \quad & \frac{1}{2}\sum_{i=0}^{N-1} \|F_i(x_i,u_i)\|_2^2 + \|F_N(x_N)\|_2^2 && \text{(2a)}\\
\text{subject to} \quad & 0 = x_0 - \bar{x}_0, && \text{(2b)}\\
& 0 = x_{i+1} - \Phi_i(x_i,u_i), \quad i = 0,\dots,N-1, && \text{(2c)}\\
& 0 \geqslant h_i(x_i,u_i), \quad i = 0,\dots,N-1, && \text{(2d)}\\
& 0 \geqslant r(x_N), && \text{(2e)}
\end{aligned}
$$

where $i = 0$ corresponds to the current time instant and with state trajectory $X = [x_0^\top,\dots,x_N^\top]^\top$ and control trajectory $U = [u_0^\top,\dots,u_{N-1}^\top]^\top$. The function $\Phi_i(x_i,u_i)$ defines the simulation of the nonlinear dynamics over one interval, starting from the state x_i and using the control values u_i. The stage cost $F_i(\cdot)$ represents the evaluation of the function F in (1a), and this is either at the corresponding shooting node or on a finer grid, for example, based on continuous output as discussed in Section 7. Direct multiple shooting addresses the problem in (2) in the full variable space (X,U).

On the other hand, a sequential approach performs the simulation separately from solving the optimization problem. One can namely obtain a reduced OCP formulation by eliminating the variables x_i using the results $X_{\text{sim}}(\bar{x}_0,U)$ of a forward simulation over the full horizon. This technique is better known as *single shooting* as presented in [27]. One can interpret multiple shooting as a lifted variant of the latter method, and the cost per Newton iteration can be made close to equal as

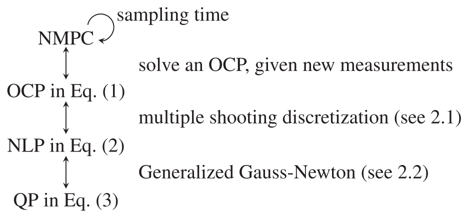
sampling time
NMPC
solve an OCP, given new measurements
OCP in Eq. (1)
multiple shooting discretization (see 2.1)
NLP in Eq. (2)
Generalized Gauss-Newton (see 2.2)
QP in Eq. (3)

Figure 1. Overview of a sequential quadratic programming-based numerical treatment of nonlinear model predictive control (NMPC). OCP, optimal control problem; NLP, nonlinear program; QP, quadratic program.

discussed in [28]. In addition, the use of multiple shooting has many advantages over single shooting. Direct multiple shooting often has better convergence properties, and it can be initialized in a more flexible way. It is also better suited for unstable systems, and the resulting algorithm can be parallelized much more easily [28].

2.2. *Generalized Gauss–Newton method*

This article restricts to the use of SQP-type algorithms to solve the NLP in (2). An important advantage over IP methods is the possibility of warm-starting them, which is particularly relevant for real-time optimal control [3]. The SQP idea is based on the quadratic model interpretation of the linearized Karush–Kuhn–Tucker system. The generalized Gauss–Newton method, for example, iterates by solving the following QP:

$$\underset{X,U}{\text{minimize}} \quad \frac{1}{2}\sum_{i=0}^{N-1}\left\| F_i^{[k]} + J_i^{[k]}\begin{bmatrix} x_i - x_i^{[k]} \\ u_i - u_i^{[k]}\end{bmatrix}\right\|_2^2 + \left\| F_N^{[k]} + J_N^{[k]}\left(x_N - x_N^{[k]}\right)\right\|_2^2 \tag{3a}$$

$$\text{subject to} \quad 0 = x_0 - \bar{x}_0, \tag{3b}$$

$$0 = x_{i+1} - \phi_i\left(x_i^{[k]}, u_i^{[k]}\right) - \left[A_i^{[k]}, B_i^{[k]}\right]\begin{bmatrix} x_i - x_i^{[k]} \\ u_i - u_i^{[k]}\end{bmatrix}, \quad i = 0,\dots,N-1, \tag{3c}$$

$$0 \geqslant h_i\left(x_i^{[k]}, u_i^{[k]}\right) + \left[C_i^{[k]}, D_i^{[k]}\right]\begin{bmatrix} x_i - x_i^{[k]} \\ u_i - u_i^{[k]}\end{bmatrix}, \quad i = 0,\dots,N-1, \tag{3d}$$

$$0 \geqslant r\left(x_N^{[k]}\right) + C_N^{[k]}\left(x_N - x_N^{[k]}\right), \tag{3e}$$

where $A_i^{[k]} = \frac{\partial \phi_i}{\partial x_i}^{[k]}$, $B_i^{[k]} = \frac{\partial \phi_i}{\partial u_i}^{[k]}$ and $C_i^{[k]} = \frac{\partial h_i}{\partial x_i}^{[k]}$, $D_i^{[k]} = \frac{\partial h_i}{\partial u_i}^{[k]}$ denote the Jacobian matrices of the constraint functions and $J_i^{[k]} = \frac{\partial F_i}{\partial (x_i,u_i)}^{[k]}$, $J_N^{[k]} = \frac{\partial F_N}{\partial x_N}^{[k]}$ the ones for the objective residual functions. In our special case of a least-squares cost, it is very common to approximate the objective as shown in (3a). It results in a Gauss–Newton Hessian approximation as presented in [29]. The matrix products $J_i^{[k]\top} J_i^{[k]}$ for $i = 0,\dots,N-1$ and $J_N^{[k]\top} J_N^{[k]}$ define the quadratic term in the objective, and all together, they approximate the full exact Hessian of the Lagrangian function in a Newton-type method [1]. This approach is computationally cheap, while it always provides a positive semidefinite Hessian approximation. The resulting convergence rate can only be shown to be linear, although the performance is often very good in practice, for example, in case of tracking NMPC. A good approximation of the exact Hessian of the Lagrangian function is obtained as long as the evaluated objective residual functions $F_i^{[k]} = F_i(x_i^{[k]}, u_i^{[k]})$ remain small.

Solving the resulting sequence of QPs and updating the current state and control trajectories given a good initial guess, the algorithm then converges to a locally optimal solution of the original NLP. Note that these principles can be extended to the framework of sequential convex programming where one preserves as much of the convexity as possible in the subproblem [30]. Each problem can directly be solved by a tailored sparsity-exploiting solver such as FORCES [6] or qpDUNES [7]. As an alternative, a condensing technique can first efficiently eliminate the state variables by use of the dynamic equality constraints [8]. The resulting small-scale QP is then solved by a dense linear algebra solver such as qpOASES [10], while the NLP iterations are still performed in the full variable space. A more elaborate discussion on when to apply condensing and when to directly solve the sparse subproblem can be found in [11]. Note that both the simulated values $\phi_i^{[k]}$ and their sensitivities $A_i^{[k]}$ and $B_i^{[k]}$ for $i = 0,\dots,N-1$ are assumed to be provided by an integrator in the course of this article.

2.3. *Real-time iterations*

In the context of real-time NMPC, the computational cost of solving a nonlinear OCP makes it hard to meet the tight timing requirements. Among other online algorithms, the RTI scheme has been proposed as a highly competitive approach [5]. A survey of online optimization algorithms for NMPC can be found in [3]. The RTI scheme, as presented in Algorithm 1, pursues to minimize the feedback time to the process. It performs only one SQP-type iteration of the generalized Gauss–Newton method per sampling time, to avoid iterating until convergence for an outdated problem. Initial value embedding, that is, the use of a constraint such as in (2b) to impose the initial state values, also plays a crucial role. The state estimate $\bar{x}_0$ enters linearly, and the optimal control solution depends differentiably on this parameter, except during active set changes. Combined with the Gauss–Newton Hessian approximation, this yields a multiplier-free, generalized tangential predictor, that is, one that can work across active set changes [3]. A final, important ingredient of the RTI scheme is its division of the computations at each sampling instant into a preparation and a feedback phase. The typically more computationally demanding preparation phase includes the problem linearization and potential condensing of the large structured QP. This preparation task can be performed before the next state estimate is available and includes most of the necessary computations. Once the new value becomes available, the feedback phase can quickly solve the prepared convex subproblem. This can drastically reduce the computational delay between receiving the new state estimate and applying the next control input to the process.

Algorithm 1 Real-Time SQP using Gauss-Newton

```
Input: initial guess $X^{[0]} = \left(x_0^{[0]}, \dots, x_N^{[0]}\right), U^{[0]} = \left(u_0^{[0]}, \dots, u_{N-1}^{[0]}\right), k = 0$
  while true do
      1. Use an integrator to obtain the simulation results $\phi_i^{[k]}$ and its first order derivatives $A_i^{[k]}$
         and $B_i^{[k]}$ for all shooting intervals $i = 0, \dots, N-1$.
      2. Prepare the QP in (3) and optionally apply the condensing algorithm from [8] to reduce
         this sparse QP to its smaller, but dense format.
      3. Wait until the current state measurement $\bar{x}_0$ arrives.
      4. Solve the QP, apply the full step $X^{[k+1]} = X^{[k]} + \Delta X$, $U^{[k+1]} = U^{[k]} + \Delta U$ and send
         the new control input $u_0^{[k+1]}$ to the process.
      5. Increment $k$ and shift the trajectories $X$ and $U$ forward in time.
  end while
```

2.4. ACADO code generation

The open-source ACADO Toolkit is written in C++ and based on the scheme in Algorithm 1; its code generation tool exports highly efficient C code for real-time optimal control [31]. An important component consists of the autogenerated integrators with efficient sensitivity propagation [15], which will be our main topic of interest throughout this article. A user-friendly MATLAB interface is available, which allows one to export, compile, and use autogenerated code for NMPC in an intuitive way and without direct interaction with C/C++ programming [17]. ACADO provides the user with easy access to the exported code to perform various NMPC simulations from MATLAB. Note that the same optimized C code could later be employed in real time on embedded control hardware. The ACADO software can be downloaded from `www.acadotoolkit.org` for reproducing all numerical results that are presented in this article.

It is important to stress that these code generation techniques are not merely restricted to problems with a short control horizon and a small dynamic system. As discussed in [11], a smart choice of the approach to solve the QP subproblem can allow one to efficiently treat short to long horizon control problems. A statement on the system dimensions that can possibly be handled by a code generation-based framework would strongly depend on the targeted application. Successful implementations of real-time feasible NMPC using ACADO can be found in [21–23, 25] or, for example, in the paper [32] on airborne wind energy, which presents total computation times below 125 ms for a control horizon of 20 intervals and the use of a nonlinear DAE model of 56 differential, 3 algebraic states, and 14 control inputs.

3. A GUIDING EXAMPLE: NONLINEAR MODEL PREDICTIVE CONTROL OF AN INVERTED PENDULUM

The guiding example that will be used throughout this article is the classical system of a pendulum mounted on top of a cart as illustrated in Figure 2. It forms an ideal tutorial example for NMPC because it is simple and intuitive but it can also exhibit rather fast dynamics and nonlinear behavior. Subsection 3.1 first describes the modeling task for this system with an alternative DAE model in Subsection 3.2. The OCP formulation can then be introduced in Subsection 3.3.

3.1. Modeling the system

The position of the cart will be denoted by w_0, and the pendulum configuration described by the angle θ, using the convention that $\theta = 0$ rad, corresponds to the pendulum hanging down in the negative y direction. The pendulum consists of a mass m attached to the cart through a massless link of length l. The position of the mass attached to the free end of the pendulum is given by

$$p = \begin{bmatrix} w_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} l\sin(\theta) + w_0 \\ -l\cos(\theta) \end{bmatrix}. \tag{4}$$

The kinetic energy of the system is given by $T = \frac{1}{2}m\dot{p}^\top\dot{p} + \frac{1}{2}M\dot{w}_0^2$, where M denotes the cart mass. The potential energy subsequently corresponds to $U = mgy_1$, where g denotes the gravitational acceleration. Using the Lagrangian formalism from [33], one can define $\mathcal{L} = T - U$ so that the equations of motion read

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{q}} - \frac{\partial\mathcal{L}}{\partial q} = F_q, \tag{5}$$

with the generalized coordinates $q = [w_0, \theta]^\top$, the generalized forces $F_q = [u, 0]^\top$, and u defined as a control input. This formulation directly yields the following implicit ODE system:

$$\begin{aligned} (M+m)\ddot{w}_0 + ml(\ddot{\theta}\cos(\theta) - \sin(\theta)\dot{\theta}^2) - u &= 0, \\ l\ddot{\theta} + \ddot{w}_0\cos(\theta) + g\sin(\theta) &= 0. \end{aligned} \tag{6}$$

By solving for the differential state derivatives $\ddot{w}_0$ and $\ddot{\theta}$, one can obtain the following explicit ODE formulation, which is mathematically equivalent:

$$\begin{aligned} \ddot{w}_0 &= \frac{ml\sin(\theta)\dot{\theta}^2 + mg\cos(\theta)\sin(\theta) + u}{M + m - m(\cos(\theta))^2}, \\ \ddot{\theta} &= -\frac{ml\cos(\theta)\sin(\theta)\dot{\theta}^2 + u\cos(\theta) + (M+m)g\sin(\theta)}{l(M + m - m(\cos(\theta))^2)}. \end{aligned} \tag{7}$$

Note that the latter equations are well defined in case that both $M \neq 0$ and $l \neq 0$.

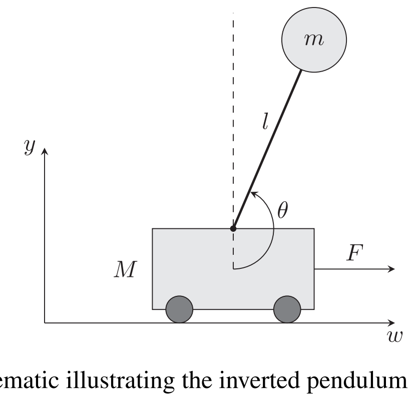

Figure 2. Schematic illustrating the inverted pendulum on top of a cart.

3.2. *An alternative differential algebraic equation model*

The same set-up can also be modeled as a DAE system. Instead of using the generalized coordinates as before, one can use $q = [w_0, w_1, y_1]$ and introduce the following constraint:

$$C(q) = \frac{1}{2}\left((w_1 - w_0)^2 + y_1^2 - l^2\right) = 0.$$

The Lagrangian is now given by $\mathcal{L} = T - U - \lambda C$, where λ is the Lagrange multiplier with respect to this constraint. The obtained DAE is of index 3, but it can be reduced to one of index 1, and the resulting equations of motion read

$$\begin{bmatrix} M & 0 & 0 & w_0 - w_1 \\ 0 & m & 0 & w_1 - w_0 \\ 0 & 0 & m & y_1 \\ w_0 - w_1 & w_1 - w_0 & y_1 & 0 \end{bmatrix} \begin{bmatrix} \ddot{w}_0 \\ \ddot{w}_1 \\ \ddot{y}_1 \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ -mg \\ u \\ -\dot{y}^2 - (\dot{w}_0 - \dot{w}_1)^2 \end{bmatrix}, \quad \begin{matrix} C(0) = 0 \\ \dot{C}(0) = 0 \end{matrix}, \tag{8}$$

where w_0, w_1, and y_1 as well as their time derivatives are differential states and λ is an algebraic state. Note that if a consistent initial state is provided, the conditions $C(0) = 0$ and $\dot{C}(0) = 0$ automatically hold and the NMPC scheme will not need to impose them.

3.3. *Optimal control problem formulation*

The targeted control task is to make the pendulum perform a full swing-up starting from the stable steady-state position described by $w_0 = 0$ m and $\theta = 0$ rad, while the reference point is unstable and corresponds to, respectively, 0 m and π rad. Let us assume that the control input and the position of the cart are both bounded, respectively, by $-20 \leqslant u \leqslant 20$ and $-2 \leqslant w_0 \leqslant 2$. Using a least-squares objective and, for example, one of the ODE models presented earlier, the OCP formulation in continuous time is the following:

$$\begin{aligned}
\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & \int_{t_0}^{t_0+T} \left(\|x(t) - x^{\text{ref}}\|_Q^2 + \|u(t) - u^{\text{ref}}\|_R^2\right) \mathrm{d}t \; + \; \|x(t_0 + T) - x^{\text{ref}}\|_P^2 && \text{(9a)} \\
\text{subject to} \quad & x(t_0) = \bar{x}_0, && \text{(9b)} \\
& \dot{x}(t) = f(x(t), u(t)), \qquad \forall t \in [t_0, t_0 + T], && \text{(9c)} \\
& -2 \leqslant w_0(t) \leqslant 2, \qquad \forall t \in [t_0, t_0 + T], && \text{(9d)} \\
& -20 \leqslant u(t) \leqslant 20, \qquad \forall t \in [t_0, t_0 + T], && \text{(9e)}
\end{aligned}$$

where the states are defined as $x = [w_0, \theta, \dot{w}_0, \dot{\theta}]^\top$, the horizon length $T = 1$ s, and the weighting matrices Q and R are defined as

$$Q = \text{diag}\left(\begin{bmatrix} 10 & 10 & 0.1 & 0.1 \end{bmatrix}\right) \text{ and } R = [0.01].$$

Note that the units are chosen consistently with the unit of measure of the states, so as to yield a dimensionless cost. To obtain a tractable problem, a multiple shooting discretization is performed using 20 intervals over the control horizon. The used NMPC sampling time is equal to $T_s = 0.05$ s to be able to deal with the fast nonlinear dynamics in the system.

3.4. *Nonlinear model predictive control using ACADO code generation*

To obtain a real-time feasible implementation of NMPC, one can export a tailored RTI scheme using the open-source ACADO code generation tool. Figure 3 presents the MATLAB code structure that can be used for this. Note that the OCP formulation in (2) is fully supported by ACADO where

```
%% -- Define the model --
DifferentialState w0 theta dw0 dtheta   % define states
Control u                               % define controls

l = 0.5;  g = 9.81;  m = 0.1;  M = 1;
expr1 = -m*cos(theta)^2 + M + m;
expr2 = m*l*sin(theta)*dtheta^2 + u;
expr3 = m*g*sin(theta);
  % define all model equations:
f = [ dot(w0); dot(theta); dot(dw0); dot(dtheta) ] == ...
    [ dw0; ...
      dtheta; ...
      (expr2 + expr3*cos(theta))/expr1; ...
     -(cos(theta)*expr2 + expr3 + M*g*sin(theta))/(l*expr1) ];

%% -- Export integrator --
sim = acado.SIMexport( 0.05 );  % sampling time Ts = 0.05
sim.setModel(f);
sim.set( 'INTEGRATOR_TYPE', 'INT_RK4' );  % integration method
sim.set( 'NUM_INTEGRATOR_STEPS', 5 );     % number of steps

sim.exportCode( 'SIM_export' );  % export code and compile
make_acado_integrator( '../acado_system' )

%% -- Export NMPC solver --
ocp = acado.OCP( 0.0,1.0,20 );  % 20 control intervals over 1s
  % least squares stage costs to define the objective:
W = diag([10 10 0.1 0.1 0.01]);
WN = diag([10 10 0.1 0.1]);
ocp.minimizeLSQ( W, [w0; theta; dw0; dtheta; u] );
ocp.minimizeLSQEndTerm( WN, [w0; theta; dw0; dtheta] );
  % state and control constraints:
ocp.subjectTo( -2.0 ≤ w0 ≤ 2.0 );
ocp.subjectTo( -20.0 ≤ u ≤ 20.0 );
ocp.setModel( f ); % constraints from dynamic model

mpc = acado.OCPexport( ocp );
mpc.set( 'QP_SOLVER', 'QP_QPOASES' );
mpc.set( 'INTEGRATOR_TYPE', 'INT_RK4' );
mpc.set( 'NUM_INTEGRATOR_STEPS', 20 );  % steps over horizon

mpc.exportCode( 'MPC_export' );  % export code and compile
make_acado_solver( '../acado_RTIstep' )

%% -- Simulation loop --
time = 0;  Ts = 0.05;  Tf = 5;
  % define reference trajectories:
input.y = repmat([0 pi 0 0 0], 20, 1);
input.yN = [0 pi 0 0].';
  % initialize state and control trajectories:
input.x = repmat([0 pi 0 0], 21, 1);
input.u = zeros(20, 1);

state_sim = zeros(4,1);
while time < Tf
    input.x0 = state_sim;  % current state values
    output = acado_RTIstep(input);  % solve OCP
      % shift state and control trajectories:
    input.x = [output.x(2:end,:); output.x(end,:)];
    input.u = [output.u(2:end,:); output.u(end,:)];

    input_sim.x = state_sim;
    input_sim.u = output.u(1,:).';
    output_sim = acado_system(input_sim);  % simulate system
    state_sim = output_sim.value;
    time = time + Ts;
end
```

Figure 3. MATLAB code example to implement nonlinear model predictive control (NMPC) using ACADO code generation.

the stage cost is defined as $F_i(x_i, u_i) = W_i^{1/2}(\tilde{F}(x_i, u_i) - y_i^{\mathrm{ref}})$, such that users can easily specify weighting matrices W_i and a reference trajectory y_i^{ref}. The following four crucial steps can be identified:

1. Define the model equations: This part of the code will be changed the most throughout the following sections, since our focus will be on the way this nonlinear system is dealt with in fast NMPC applications.
2. Simulation routine: An integrator is generated to accurately simulate the behavior of the process after applying the next control inputs. Although only ACADO is being used here, this component can be replaced by any available simulation environment.
3. NMPC export: The OCP needs to be formulated in ACADO syntax, including the dynamics, the objective functions, and constraints, so that a corresponding solver can be exported. Various options can be used to customize the code-generated algorithm.
4. Simulation: The exported solver can be used to run various closed-loop NMPC simulations while keeping the efficiency of the optimized C code for the relevant computations.

ACADO keywords can be used to define the various differential states and controls. They are used to build all symbolic expressions that appear in the differential equations. After defining the model in Figure 3, the code exports the four-stage ERK method of order 4 (ERK4) as a stand-alone integrator to simulate the system. It is specified to perform five integration steps to simulate the model over 0.05 s, resulting in a relatively high accuracy. The `SIMexport` module generates the optimized C code into the folder called 'SIM_export'. Notice that there are options to customize the integrator and to include an efficient propagation of sensitivities. This turns it into a powerful tool to prototype new algorithms as illustrated in [17] and also further in Section 7.2. The third step involves the use of the `OCP` and `OCPexport` module, which generates the RTI-based solver for NMPC. It allows the user to specify the embedded components such as the integrator and the convex solver. In the code from Figure 3, the same ERK method is used and combined with a condensing technique and qpOASES as the underlying solver. As mentioned earlier, it is possible to alternatively employ a structure-exploiting solver such as FORCES or qpDUNES. The self-contained C code is exported into the specified 'MPC_export' folder. Finally, the presented code also automatically generates MEX functions, which can be used to perform

- an evaluation of the model's right-hand side and its derivatives,
- a call to the integrator and its optional sensitivity propagation, and
- the RTI scheme to solve the user-provided OCP instance.

Using this functionality, one can rather intuitively perform various NMPC simulations by consecutively calling the OCP solver and the integrator in a loop.

4. EXPLICIT INTEGRATION METHODS

Let us first present the case of having an OCP formulation where the dynamics are described by an explicit ODE system. In that case, an explicit integration method can be used to solve the following initial value problem:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u(t)), \\ x(0) &= x_0. \end{aligned} \tag{10}$$

This section briefly covers ERK methods, which are first introduced in Subsection 4.1. An overview on techniques for first-order sensitivity analysis is subsequently provided in Subsection 4.2. Finally, an illustration of the resulting NMPC implementation and its performance will be presented in Subsection 4.3.

4.1. Explicit Runge–Kutta methods

Runge–Kutta (RK) methods form an important class of one-step methods, which are attractive for multiple shooting because they do not require any start-up procedure. Detailed information on the derivation and properties of these methods can be found in [34]. One can use the following s-stage RK method to integrate the ODE system in (10) from time t_n to $t_n + h$:

$$\begin{aligned} k_i &= f(t_n + c_i h, x_n + h \sum_{j=1}^{s} a_{ij} k_j, u_n), \quad \text{for } i = 1, \ldots, s, \\ x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{11}$$

where the values b_i denote the weights, c_i are the coefficients defining the s stages, and a_{ij} are called the internal coefficients. For an ERK method, the coefficient matrix $A = [a_{ij}]$ must be strictly lower triangular so that the variables k_i are defined explicitly in (11). The simplest method is the well-known forward Euler method. It is usually more efficient to use a higher-order method such as the methods of Runge and Heun or the popular four-stage ERK method of order 4 (ERK4) [34]. It is important to note that explicit methods should typically not be used in case of a stiff problem, as detailed in [35].

4.2. *Sensitivity analysis*

A straightforward way of computing first-order sensitivity information is to simulate the ODE system, extended with the following forward variational differential equations (VDEs):

$$\begin{aligned} \dot{S}_x(t) &= \frac{\partial f(t, x(t), u(t))}{\partial x} S_x(t), \\ \dot{S}_u(t) &= \frac{\partial f(t, x(t), u(t))}{\partial x} S_u(t) + \frac{\partial f(t, x(t), u(t))}{\partial u}, \\ [\, S_x(0) \mid S_u(0)\,] &= [\, \mathbb{1} \mid \mathbb{0}\,], \end{aligned} \tag{12}$$

where the sensitivity matrices are defined as $S_x(t) = \frac{\partial x(t)}{\partial x_0}$ and $S_u(t) = \frac{\partial x(t)}{\partial u}$ and u is assumed to consist of all control values on which the simulated state depends. The initial value $\mathbb{1}$ denotes an identity matrix and $\mathbb{0}$ contains only zeros. Targeting real-time applications, a fixed step size and order is assumed for the code-generated integrator [13]. In case of an explicit method, the simulation of these VDEs is then equivalent to the use of internal numerical differentiation where one would differentiate the method itself [29]. Both approaches therefore automatically lead to an efficient scheme in combination with algorithmic differentiation (AD) to evaluate the derivatives [36]. Note that each forward approach allows an alternative where one performs a backward propagation instead, for example, using the reverse mode of AD [37]. This can be more efficient in case of many parameters with respect to which the sensitivities are needed. Because the presented methods are later extended with extra outputs, a forward scheme will be preferred throughout the remainder of this article.

4.3. *Application: nonlinear model predictive control on an inverted pendulum*

As mentioned earlier, efficient C code for NMPC can be automatically generated using ACADO from MATLAB. Similar to before, let us employ the four-stage ERK4 with a step size of 0.05 s. The expressions for the explicit ODE system from (7) can again be formulated as in the lines 1–15 of the code in Figure 3. The closed-loop NMPC behavior for the full swing-up within 4 s is shown in Figure 4. The average computation time for different implementations of this scheme can be found in Table I. The integration time includes the simulation of the model over all 20 shooting intervals together with its sensitivity propagation. It can be observed from this table that the integrator and QP solver often form the computationally most expensive ingredients for the RTI scheme. The sum of the integration and QP solution times for, for example, ACADO using FORCES is 189 μs, which is close to the total computation time of 199 μs per step. Note that FORCES does not appear to be the ideal choice here as the embedded QP solver. The use of qpDUNES results in faster code

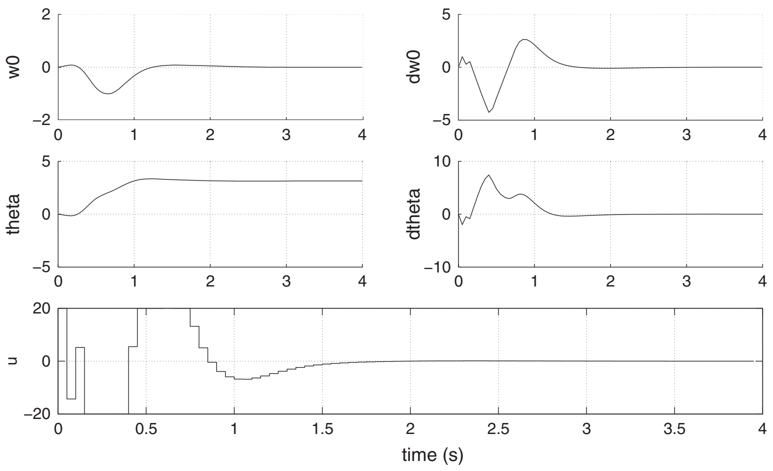

Figure 4. Illustration of the closed-loop nonlinear model predictive control behavior for the original optimal control problem formulation.

Table I. Average computation time per RTI step, interfacing an ACADO integrator and QP solver.

	`quadprog–ode45` (μs)	FORCES (μs)	qpDUNES (μs)	qpOASES (μs)
$N = 20$				
Integration (ERK4)	60680	34	34	34
QP solver	61504	155	52	28
Condensing	–	–	–	12
ACADO total RTI	–	199	97	80
$N = 60$				
ACADO total RTI		745	312	407

ERK4, explicit Runge–Kutta method of order 4; QP, quadratic program; RTI, real-time iteration.

in this case, but condensing combined with qpOASES provides the most efficient implementation with only 80 μs per time step. On longer horizons, a sparsity-exploiting solver such as FORCES or qpDUNES can be shown to eventually become faster than the condensing-based approach as discussed in [11]. The timing results from a simulation using $N = 60$ control intervals over a horizon, which is also three times longer, are included in Table I for illustrative purposes. In this specific case, the qpDUNES-based implementation becomes the most efficient one. The table also presents the average computation time when using `quadprog` and `ode45`, respectively, to solve the QP and to perform integration and sensitivity computations. The difference between an ACADO-exported integrator and `ode45` is quite noticeable, mostly because of using code generation and fixing the step size. The propagation of sensitivities with `ode45` is performed using the system extended with the VDEs from (12).

5. IMPLICIT INTEGRATION METHODS

A quite pragmatical definition of stiffness is when the dynamic equations are 'such that certain implicit methods perform better, usually tremendously better, than explicit ones' [35]. Many real-world problems are observed to be stiff [38]; that is, it is rather important to include implicit schemes in this discussion. Moreover, the initial value problem formulation from (10) can be generalized to

$$\begin{aligned} 0 &= f(t, \dot{x}(t), x(t), z(t), u(t)), \\ x(0) &= x_0, \end{aligned} \tag{13}$$

with $x(t)$ a vector of n_x differential states, $\dot{x}(t)$ the corresponding time derivatives, $z(t)$ a vector of n_z algebraic states, and $u(t)$ a vector of n_u control inputs. This covers both the implicit ODE model from (6) and the index-1 DAE model from (8). The only assumption is that the Jacobian matrix $\frac{\partial f}{\partial(z,\dot{x})}$ is invertible. Subsection 5.1 briefly describes the implementation of implicit RK (IRK) methods, while their tailored sensitivity analysis is treated in Subsection 5.2. The effect of the model formulation on the NMPC scheme for our guiding example is briefly investigated in Subsection 5.3.

5.1. *Implicit Runge–Kutta methods*

Because of their high order of accuracy and good stability properties, one is often interested in A-stable IRK methods such as the Gauss or Radau IIA methods [35]. The RK scheme can be directly applied to the fully implicit DAE system from (13) as follows:

$$\begin{aligned} 0 &= f\left(t_n + c_i h, k_i, x_n + h\sum_{j=1}^{s} a_{ij} k_j, Z_i, u_n\right), \; i = 1, \dots, s, \\ x_{n+1} &= x_n + h \sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{14}$$

where the coefficients b_i, c_i, and $a_{i,j}$ define the method as before, Z_i denotes the stage values of the algebraic states, and k_i denotes the stage values of the differential state derivatives. The result is a nonlinear system of $s(n_x + n_z)$ equations that must be solved using a Newton-type method. Targeting real-time applications, not only the step size and order of the method but also the number of Newton iterations L is assumed to be kept fixed. Given the results from a previous integration step, one can construct a reasonable initialization for the next stage values. The number of iterations L should be chosen in a conservative way, considering the applications for which the resulting scheme will be employed [15]. Let us write the nonlinear system as $G(r_n, K) = 0$ where $r_n = (x_n, u_n)$; that is, the Newton iterations are

$$K^{[i]} = K^{[i-1]} - M^{-1} G\left(r_n, K^{[i-1]}\right), \; i = 1, \dots, L, \tag{15}$$

where M is an approximation of the Jacobian $\frac{\partial G}{\partial K}$ and $K = (k_1, \dots, k_s, Z_1, \dots, Z_s)$. The evaluation of derivatives is done efficiently using AD, and for example, a custom linear solver can be used, based on the lower upper (LU) decomposition of the matrix M. Note that the iterations itself are computationally cheaper than this matrix factorization, motivating the reuse of a Jacobian approximation [15].

5.2. *Sensitivity propagation*

For similar reasons as before, forward techniques for sensitivity propagation will be targeted. Following a *differentiate-then-integrate* approach, it would again be possible to extend the dynamics with a system of sensitivity equations. This description is insufficient in case of implicit methods because the resulting efficiency will strongly depend on how the augmented system is treated. A detailed discussion on techniques of forward sensitivity propagation can be found in [16], including internal numerical differentiation for implicit methods. The conclusion there is that an efficient approach is to directly apply the implicit function theorem to the system $G(r_n, K) = 0$, resulting in

$$\frac{dK}{dr_n} = -\frac{\partial G}{\partial K}^{-1} \frac{\partial G}{\partial r_n}, \tag{16}$$

where the Jacobian $\frac{\partial G}{\partial K}$ needs to be evaluated and factorized. But this factorization can be reused in the Newton iterations of the next integration time step, as proposed and motivated by [15, 16]. Algorithm 2 compactly describes the resulting implementation of an IRK method with tailored sensitivity propagation. Only one LU factorization is needed per integration step.

Algorithm 2 The implementation of one IRK integration step

```
Input: $r_n$, initial $K^{[0]}$, LU factorization of $M$
Output: $M$, $(x_{n+1}, \frac{\partial x_{n+1}}{\partial r_n})$ and $\left(z_n, \frac{\partial z_n}{\partial r_n}\right)$
1: for $i = 1 \rightarrow L$ do
2:     $K^{[i]} \leftarrow K^{[i-1]} - M^{-1} G\left(r_n, K^{[i-1]}\right)$
3: end for
4: $x_{n+1} \leftarrow x_n + h \sum_{i=1}^{s} b_i k_i$
5: $z_n \leftarrow \sum_{i=1}^{s} l_i(0) Z_i$   with  $l_i(t) = \prod_{j \neq i} \frac{t-c_j}{c_i-c_j}$
6: $M \leftarrow \frac{\partial G}{\partial K}\left(r_n, K^{[L]}\right)$
7: compute $\frac{dK}{dr_n}$ using $M$ in Eq. (16)
8: $\frac{\partial x_{n+1}}{\partial r_n} \leftarrow \frac{\partial x_n}{\partial r_n} + h \sum_{i=1}^{s} b_i \frac{dk_i}{dr_n}$
9: $\frac{\partial z_n}{\partial r_n} \leftarrow \sum_{i=1}^{s} l_i(0) \frac{dZ_i}{dr_n}$
```

5.3. Application: nonlinear model predictive control using a differential algebraic equation model

Regarding our guiding example, one can also use the implicit ODE in (6) or the DAE model in (8) for NMPC. The ACADO formulation of the implicit DAE system looks as follows:

```
DifferentialState w0 w1 y1 dw0 dw1 dy1
AlgebraicState lambda
Control u

l = 0.5;  g = 9.81;  m = 0.1;  M = 1;
expr1 = (dot(dw0) - dot(dw1))*(w0 - w1);
expr2 = lambda*(w0 - w1);
expr3 = (dw0 - dw1)^2;

f = [ 0 == dw0 - dot(w0);
      0 == dw1 - dot(w1);
      0 == dy1 - dot(y1);
      0 == M*dot(dw0) - u + expr2;
      0 == m*dot(dw1) - expr2;
      0 == m*dot(dy1) + m*g + lambda*y1;
      0 == dy1^2 + dot(dy1)*y1 + expr3 + expr1 ];
```

This code can be used as before to generate an IRK method with tailored sensitivity propagation or to export a full RTI scheme. The average computation time for one RTI step is presented in Table II, using each of the three possible model formulations. The ACADO-generated scheme uses condensing combined with qpOASES, and the results also show the time spent in integration and sensitivity propagation. The same IRK method is used for all three model formulations, namely a Gauss method of order 2 (IRK method of order 2) with a step size of 0.05 s. The different NMPC implementations seem to result in a rather similar execution time; that is, there is little to no overhead corresponding to treating an implicit ODE or even DAE system instead of the simple explicit model. An implicit ODE or a DAE formulation can sometimes even lead to faster computation times because of simpler expressions in the model, which is observed for the implicit ODE in this case. Note that the DAE system consists of seven instead of four states (six differential and one algebraic state), while its corresponding execution time is comparable to the others.

6. MODELS WITH LINEAR SUBSYSTEMS

Many dynamic systems are described by a set of nonlinear differential equations with possibly a few algebraic states. One would often identify coupled subsystems that can be simulated in separate stages. In the special case that such a subsystem is linear, this can be additionally exploited. Subsection 6.1 presents a three-stage model formulation, which has been observed to appear rather often

Table II. Average computation time for NMPC using different model formulations in ACADO.

	Explicit ODE (7) (μs)	Implicit ODE (6) (μs)	Implicit DAE (8) (μs)
Integration (IRK2)	24	22	32
Total RTI (qpOASES)	60	58	85

DAE, differential algebraic equation; IRK2, implicit Runge–Kutta method of order 2; NMPC, nonlinear model predictive control; ODE, ordinary differential equation; RTI, real-time iteration.

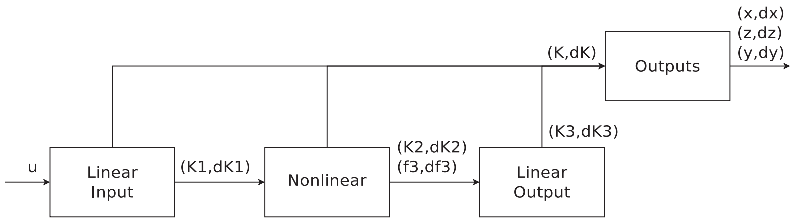

Figure 5. Schematic illustrating the workflow in structure-exploiting Runge–Kutta integrators.

in practice. The impact on the implementation of autogenerated RK methods is discussed in Subsection 6.2. The specific structure is eventually illustrated to serve as a tool to implement popular NMPC formulations in Subsection 6.3.

6.1. *The three stage structure*

Let us consider the following three-stage model structure from [18]:

$$C_1\dot{x}^{[1]} = A_1x^{[1]} + B_1u, \tag{17a}$$

$$0 = f_2\left(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u\right), \tag{17b}$$

$$C_3\dot{x}^{[3]} = A_3x^{[3]} + f_3\left(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u\right), \tag{17c}$$

with matrices A_1, B_1, and A_3, invertible matrices C_1 and C_3, and nonlinear functions f_2 and f_3. The main assumption is that the Jacobian matrix $\frac{\partial f_2}{\partial(z,\dot{x}^{[2]})}$ is invertible; that is, the second subsystem represents an implicit DAE of index 1. Linear input (17a) and output (17c) systems can, for example, originate from partially linear dynamics or from filtering actions, respectively, before and after the nonlinear dynamics. In case that $A_3 = 0$ and C_3 is an identity matrix, (17c) reduces to

$$\dot{x}^{[3]} = f_3\left(x^{[1]}, x^{[2]}, \dot{x}^{[1]}, \dot{x}^{[2]}, z, u\right) \tag{18}$$

which are better known as quadrature states [39]. They are typically used to formulate objective and constraint functions, similar to how the more general linear input and output states can be used.

6.2. *Structure exploiting Runge–Kutta methods*

It is possible to exploit the specific structure when applying an RK method to the system in (17), resulting in a workflow such as depicted in Figure 5. The different subsystems are treated separately, and the results are collectively used to generate the desired outputs and their sensitivities. Note that the Jacobian matrix $\frac{dK}{dr_n}$ now has a clear sparsity structure

$$\frac{dK}{dr_n} = \begin{pmatrix} \frac{dK_1}{dx^{[1]}} & 0 & 0 & \frac{dK_1}{du} \\ \frac{dK_2}{dx^{[1]}} & \frac{dK_2}{dx^{[2]}} & 0 & \frac{dK_2}{du} \\ \frac{dK_3}{dx^{[1]}} & \frac{dK_3}{dx^{[2]}} & \frac{dK_3}{dx^{[3]}} & \frac{dK_3}{du} \end{pmatrix}, \tag{19}$$

where K_1, K_2, and K_3 denote the stage values corresponding to the three subsystems. Because of linearity, the values $\frac{dK_1}{dx^{[1]}}$, $\frac{dK_1}{du}$, and $\frac{dK_3}{dx^{[3]}}$ are constant and can be precomputed in a code generation framework. In case of IRK methods, the speedup to be made is typically much larger. Linear subsystems namely allow the inverse of the constant Jacobian to be precomputed offline, which reduces the cost of computing stage values to merely one matrix-vector multiplication [18].

6.3. Application: frequency-weighted nonlinear model predictive control

For our simple guiding example, the model dynamics in (6) are coupled, and there is no linear subsystem to be identified. In more realistic cases, these subsystems appear quite naturally as is shown by the control examples in [18]. Let us illustrate possible uses of both the linear input and output systems to efficiently implement popular NMPC formulations in practice.

Linear input system Instead of controlling the input u for the inverted pendulum directly, one would often use the control rate u_R and include the dynamic equation $\dot{u} = u_R$. Extra constraints on the rate of change of the input u can then be included in the OCP formulation. The resulting effect is that the NMPC controller will typically behave less aggressively and therefore less nervously in the presence of noise [14].

Linear output system Everything that can be achieved using quadrature states can be done similarly with a linear output system. But it can, for example, also be used to implement first-order high-pass filtering on a state to extra penalize its high frequency content as illustrated in [18]. Let us carry this out for the angle θ from our guiding example, which results in

$$\dot{\theta}^{\mathrm{HP}} = \dot{\theta} - \omega_C \theta^{\mathrm{HP}}, \tag{20}$$

where $\omega_C = 2\pi f_C$ and f_C is the cutoff frequency. The implicit ODE model for the inverted pendulum on a cart can now easily be extended with both the linear input and output systems:

```
DifferentialState u w0 theta dw0 dtheta theta_HP
Control uR

A1 = 0;  B1 = 1;                 % linear input
f2 = ... ;                       % nonlinear system
A3 = -6*pi;  f3 = dot(theta);    % linear output
sim = acado.SIMexport( 0.05 );
sim.setLinearInput( A1, B1 );
sim.setModel( f2 );
sim.setLinearOutput( A3, f3 );
```

The updated OCP formulation includes a constraint $-100 \leqslant u_R \leqslant 100$ and additional weights on both u_R and θ^{HP} in the objective. The average computation time per RTI step using ACADO is presented in Table III. Unlike the results in Table II, a Gauss method of order 4 (IRK method of order 4) instead of order 2 has been used for illustrative purposes. The table includes the computation time for the original and extended NMPC scheme, comparing its implementation both with and without structure exploitation. Note that the use of linear subsystems allowed us to implement this more practical scheme at the cost of merely 132 μs instead of the original 116 μs per step. A comparison between the NMPC behavior using either the original or the frequency-weighted formulation is shown in Figure 6. It can be observed that the controller is less aggressive and the trajectories appear slightly smoother.

7. CONTINUOUS OUTPUT

To efficiently define certain objective or constraint functions in an OCP formulation, additionally simulated outputs are often needed. The idea of *continuous output* is to be able to evaluate such expressions on an arbitrarily fine grid, which is independent of the integration grid. Note that this implies that the integrator has to deliver their sensitivity information. A possible use case has been

Table III. Average computation time for frequency-weighted NMPC in ACADO.

	Original NMPC using ODE in (6) (μs)	Frequency-weighted Unstructured (μs)	Frequency-weighted Linear subsystems (μs)
Integration (IRK4)	60	105	65
Total RTI (qpOASES)	116	175	132

IRK4, implicit Runge–Kutta method of order 4; NMPC, nonlinear model predictive control; ODE, ordinary differential equation; RTI, real-time iteration.

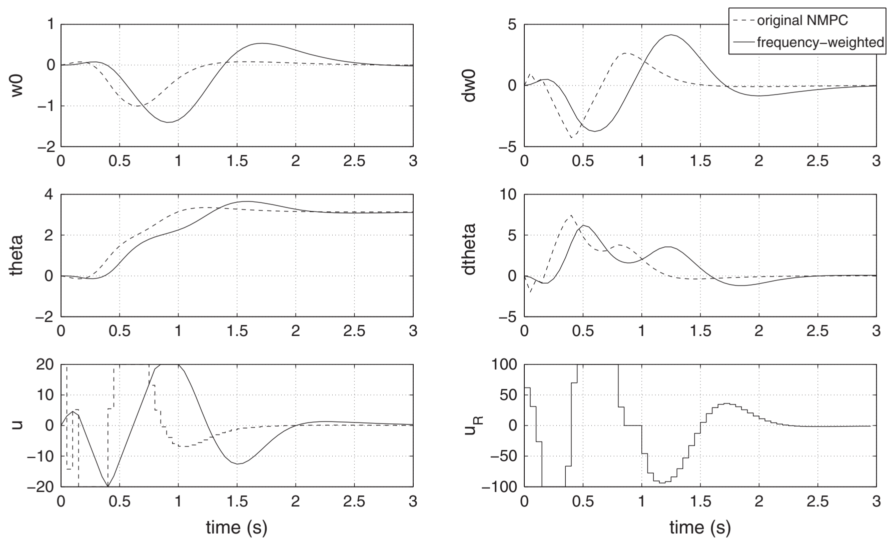

Figure 6. Simulation results for nonlinear model predictive control using either the original or the frequency-weighted formulation.

presented in [17], where multirate measurements for moving horizon estimation were efficiently incorporated in a real-time feasible scheme. Other control-related ideas could be to approximate a least-squares integral or to define certain constraints on a rather fine grid. Subsection 7.1 briefly presents the support of continuous output for a specific family of IRK methods. An interesting NMPC application is introduced in Subsection 7.2, where an infinite horizon closed-loop costing scheme will be implemented in an efficient way.

7.1. Collocation methods

Collocation methods are a specific family of IRK methods for which their continuous extension arises quite naturally. Using the collocation variables $K = (k_1, \dots, k_s, Z_1, \dots, Z_s)$, a continuous time polynomial is defined for the states $x(t)$, $\dot{x}(t)$, and $z(t)$ over the interval $t \in [t_n, t_{n+1}]$

$$\begin{aligned} x(t_n + ch) &\approx x_n + h \sum_{i=1}^{s} k_i \int_0^c l_i(\tau)\,\mathrm{d}\tau, \\ \dot{x}(t_n + ch) &\approx \sum_{i=1}^{s} l_i(c) k_i, \\ z(t_n + ch) &\approx \sum_{i=1}^{s} l_i(c) Z_i, \end{aligned} \tag{21}$$

where $0 \leqslant c \leqslant 1$ and $l_i(t) = \prod_{j \neq i} \frac{t - c_j}{c_i - c_j}$ are the Lagrange interpolating polynomials. In case of an IRK method of order p, the order of this approximation is $p^* = \min(p, s+1)$ for $x(t)$ and $p^* - 1$ for $\dot{x}(t)$ and $z(t)$ [17]. It is generally possible to construct interpolants for all RK methods [40].

7.2. Application: nonlinear model predictive control using infinite horizon closed-loop costing

Until now, the issue of nominal closed-loop stability of the implemented NMPC scheme has not been addressed although it forms a crucial property. Instead of determining a suitable terminal region or sufficiently prolonging the control horizon [41], let us briefly look into the technique of *infinite horizon closed-loop costing* [42, 43]. The scheme is based on a local control law and uses an infinite prediction horizon in which the state and input constraints are still imposed. A tractable approximation of the infinite horizon cost associated with the local controller is used as a penalty on the terminal state. Stability of the closed-loop system can then be proven in a rigorous way [43]. To formulate a practical scheme, the prediction horizon can be truncated without losing this stability guarantee as is the topic of discussion in [42]. Furthermore, the path constraints will only be imposed at specific time points. The main advantage of introducing the prediction horizon T_p is that it allows us to enlarge the region of attraction around the reference point without increasing the control horizon T_c, that is, with the same number of decision variables [44].

Let us adapt the OCP formulation from (9) accordingly:

$$\begin{aligned}
\underset{x(\cdot),u(\cdot)}{\text{minimize}} \quad & \int_{t_0}^{t_0+T} \left(\|x(t)-x^{\text{ref}}\|_Q^2 + \|u(t)-u^{\text{ref}}\|_R^2\right) \mathrm{d}t && \text{(22a)}\\
\text{subject to} \quad & x(t_0) = \bar{x}_0, && \text{(22b)}\\
& \dot{x}(t) = f(x(t),u(t)), \qquad \forall t \in [t_0, t_0+T], && \text{(22c)}\\
& u(t) = u_{\text{LQR}}(x(t)), \qquad \forall t \in [t_0+T_c, t_0+T], && \text{(22d)}\\
& -\bar{h} \leqslant h(t) \leqslant \bar{h}, \qquad \forall t \in [t_0, t_0+T], && \text{(22e)}
\end{aligned}$$

where $T = T_c + T_p$, the function h is defined as $h(t) = [w_0(t), u(t)]^\top$, and the bounding values are $\bar{h} = [2, 20]^\top$. The unstable steady state is denoted by $(x_{\text{ss}}, u_{\text{ss}})$, such that the Linear Quadratic Regulator (LQR) which locally stabilizes this reference point can be defined as $u_{\text{LQR}} = u_{\text{ss}} - K_p(x - x_{\text{ss}})$ where $K_p \in \mathbb{R}^{n_u \times n_x}$. The corresponding discrete time OCP formulation reads

$$\begin{aligned}
\underset{X,U}{\text{minimize}} \quad & \frac{1}{2}\sum_{i=0}^{N_c-1} \|x_i - x^{\text{ref}}\|_Q^2 + \|u_i - u^{\text{ref}}\|_R^2 \; + \; \|g(x_{N_c}) - g^{\text{ref}}\|_W^2 && \text{(23a)}\\
\text{subject to} \quad & 0 = x_0 - \bar{x}_0, && \text{(23b)}\\
& 0 = x_{i+1} - \Phi_i(x_i, u_i), \qquad i = 0,\dots,N_c-1, && \text{(23c)}\\
& -\bar{h} \leqslant h(x_i) \leqslant \bar{h}, \qquad i = 0,\dots,N_c-1, && \text{(23d)}\\
& -\bar{h} \leqslant h_j(x_{N_c}) \leqslant \bar{h}, \qquad j = 0,\dots,N_p, && \text{(23e)}
\end{aligned}$$

where the functions $h_j(x_{N_c})$ and $g(x_{N_c})$ are defined by a forward simulation over the prediction horizon using the linear control law, $g^{\text{ref}} = [x^{\text{ref}}; u^{\text{ref}}; \dots] \in \mathbb{R}^{N_p(n_x+n_u)}$ denotes the corresponding reference values, and $W = \text{diag}(Q, R, \dots) \in \mathbb{R}^{N_p(n_x+n_u)\times N_p(n_x+n_u)}$ is a block-diagonal weighting matrix. Let us consider a control horizon of $N_c = 10$ and a prediction horizon of $N_p = 20$, that is, respectively, $T_c = 0.5$s and $T_p = 1.0$ s, using a discretization with the same sampling time $T_s = 0.05$ s in both. One can generate an ACADO integrator, dedicated to perform the simulation over the prediction horizon. The goal is to take rather large integration steps but to use the continuous output feature to efficiently define the extra constraints and the terminal cost.

```
Ts = 0.05;  Np = 20;
sim = acado.SIMexport( Np*Ts );  % Tp = Np*Ts
sim.setModel( f );
  % output states and controls at Np equidistant points:
sim.addOutput( [w0; theta; dw0; dtheta; u], Np );
sim.set( 'INTEGRATOR_TYPE',            'INT_IRK_GL4'   );
sim.set( 'NUM_INTEGRATOR_STEPS',       4               );
```

This will be compared with the original NMPC scheme with $N_c = 20$ control intervals and no prediction horizon (case 1). Note that it becomes difficult to reduce the size of the horizon further and

Table IV. Average computation time for approximate infinite horizon NMPC.

	Original NMPC (6) ($N_c = 20$, $N_p = 0$)	Infinite horizon costing ($N_c = 10$, $N_p = 20$)	
	Case 1 (μs)	Case 2 (μs)	Case 3 (μs)
Integration (IRK4)	60	30	30
Prediction	–	60	14
QP (FORCES)	155	114	114
Sum	215	204	158

IRK4, implicit Runge–Kutta method of order 4; NMPC, nonlinear model predictive control; QP, quadratic program.

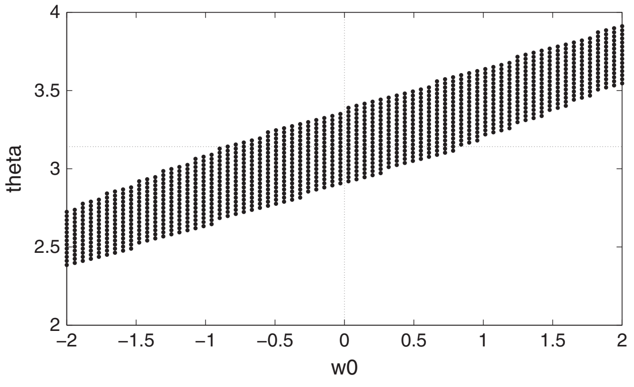

Figure 7. This figure shows the implicitly defined terminal region by nonlinear model predictive control with closed-loop costing where feasible points in (w_0, θ) space are dotted and infeasible points left blank (other two states are kept zero).

to still obtain a stable controller in this case. The resulting average computation time per RTI step is presented in Table IV using a Gauss method of order 4 and step size 0.05 s, which is interfaced to a FORCES solver. The implementation using continuous output (case 3) is much faster than the one without (case 2), where a separate integration step needs to be performed for each interval. Note that the latter scheme delivers a higher integration accuracy over the prediction horizon, although it is not necessary for our example. Integrators with continuous output provide the user with the flexibility needed to compute all results in an efficient way and with the desired precision. It is interesting that this approach can be interpreted as one that implicitly generates a terminal region as illustrated in Figure 7. It shows how much the inverted pendulum is allowed to be tilted and in which direction depending on the position of the cart, such that the LQR controller can still stabilize it under the restrictions of our system.

8. CONCLUSIONS AND FURTHER DEVELOPMENTS

This article has summarized recent algorithmic progress on autogenerated integration methods and illustrated ways to employ them for NMPC. Code-generated integrators with tailored sensitivity propagation can form a powerful tool. Selecting a suitable method and exploiting its features can allow a range of new NMPC formulations and applications to become real-time and feasible. Based on the ACADO code generation tool from MATLAB, a tutorial-style approach has been adopted using a simple guiding example that consists of an inverted pendulum on a cart. In addition, the potential to easily prototype new algorithms was briefly illustrated. The standalone implicit integrators with continuous output allowed us to efficiently implement NMPC with closed-loop costing.

These ideas can be extended to support tailored, embedded algorithms for distributed as well as robust formulations of optimal control. Such possible extensions as well as novel techniques for efficient sensitivity propagation are among the topics of ongoing research.

ACKNOWLEDGEMENTS

The authors would like to thank all reviewers for their valuable comments, which helped to improve this article. This research was supported by the following institutions: KUL: Optimization in Engineering Center OPTEC (PFV/10/002), GOA/10/09 and GOA/10/11; Flemish Government: IOF/KP/SCORES4CHEM; FWO: PhD/postdoc grants and projects: G.0320.08 and G.0377.09; IWT: PhD grants and projects: SBO LeCoPro; Belgian Federal Science Policy Office: IUAP P7 (DYSCO, Dynamical systems, control and optimization, 2012–2017); and the European Union's Seventh Framework Programme: EMBOCON (248940), SADCO (264735), TEMPO (607957), Eurostars SMART, and ERC HIGHWIND (259166). The first author holds a PhD fellowship of the Research Foundation – Flanders (FWO).

REFERENCES

1. Nocedal J, Wright SJ. *Numerical Optimization* (2nd edn), Springer Series in Operations Research and Financial Engineering. Springer: New York, 2006.
2. Wächter A, Biegler LT. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming* 2006; **106**(1):25–57.
3. Diehl M, Ferreau HJ, Haverbeke N. In *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. Springer: Berlin Heidelberg, 2009; 391–417.
4. Kirches C, Wirsching L, Sager S, Bock HG. Efficient numerics for nonlinear model predictive control. In *Recent Advances in Optimization and Its Applications in Engineering*, Diehl M, Glineur FF, Michiels JW (eds). Springer: Berlin Heidelberg, 2010; 339–357.
5. Diehl M, Bock HG, Schlöder JP. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization* 2005; **43**(5):1714–1736.
6. Domahidi A, Zgraggen A, Zeilinger MN, Morari M, Jones CN. Efficient interior point methods for multistage problems arising in receding horizon control. *IEEE Conference on Decision and Control (CDC)*, Maui, HI, USA, 2012; 668–674.
7. Frasch JV, Sager S, Diehl M. A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computations* 2013. Optimization online: http://www.optimization-online.org/DB_HTML/2013/11/4114.html.
8. Andersson J. A general-purpose software framework for dynamic optimization. *PhD Thesis*, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, 2013.
9. Frison G, Jorgensen J. A fast condensing method for solution of linear-quadratic control problems. *Proceedings of the 52nd IEEE Conference on Decision and Control*, Florence, Italy, 2013; 7715–7720.
10. Ferreau HJ. An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. *Master's Thesis*, University of Heidelberg, Baden-Württemberg, Germany, 2006.
11. Vukov M, Domahidi A, Ferreau HJ, Morari M, Diehl M. Auto-generated algorithms for nonlinear model predictive control on long and on short horizons. *Proceedings of the 52nd Conference on Decision and Control (CDC)*, Florence, Italy, 2013; 5113–5118.
12. Ferreau HJ, Kraus T, Vukov M, Saeys W, Diehl M. High-speed moving horizon estimation based on automatic code generation. *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012)*, Maui, Hawaii, 2012; 687–692.
13. Houska B, Ferreau HJ, Diehl M. An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica* 2011; **47**(10):2279–2285.
14. Vukov M, Van Loock W, Houska B, Ferreau HJ, Swevers J, Diehl M. Experimental validation of nonlinear MPC on an overhead crane using automatic code generation. *The 2012 American Control Conference*, Montreal, Canada, 2012; 6264–6269.
15. Quirynen R, Vukov M, Diehl M. Auto generation of implicit integrators for embedded NMPC with microsecond sampling times. In *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference,* Noordwijkerhout, Netherlands, 2012; 175–180.
16. Quirynen R. Automatic code generation of implicit Runge-Kutta integrators with continuous output for fast embedded optimization. *Master's Thesis*, University of Leuven, Belgium, KU Leuven, 2012.
17. Quirynen R, Gros S, Diehl M. Fast auto generated ACADO integrators and application to MHE with multi-rate measurements. *Proceedings of the European Control Conference*, Zurich, Switzerland, 2013; 3077–3082.
18. Quirynen R, Gros S, Diehl M. Efficient NMPC for nonlinear models with linear subsystems. *Proceedings of the 52nd IEEE Conference on Decision and Control*, Florence, Italy, 2013; 5101–5106.

19. Ohtsuka T, Kodama A. Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers* 2002; **38**(7):617–623.
20. Mattingley J, Boyd S. *Convex Optimization in Signal Processing and Communications*, Chapter Automatic Code Generation for Real-time Convex Optimization. Cambridge University Press, 2009.
21. Frasch JV, Gray AJ, Zanon M, Ferreau HJ, Sager S, Borrelli F, Diehl M. An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. *Proceedings of the European Control Conference*, Zurich, Switzerland, 2013; 4136–4141.
22. Gros S, Zanon M, Vukov M, Diehl M. Nonlinear MPC and MHE for mechanical multi-body systems with application to fast tethered airplanes. *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference*, Noordwijkerhout, The Netherlands, 2012; 86–93.
23. Debrouwere F, Vukov M, Quirynen R, Diehl M, Swevers J. Experimental validation of combined nonlinear optimal control and estimation of an overhead crane. *Proceedings of the 19th World Congress of the International Federation of Automatic Control*, Cape Town, South Africa, 2014; 9617–9622.
24. Geebelen K, Vukov M, Wagner A, Ahmad H, Zanon M, Gros S, Vandepitte D, Swevers J, Diehl M. An experimental test setup for advanced estimation and control of an airborne wind energy systems. In *Airborne Wind Energy*, Ahrens U, Diehl M, Schmehl R (eds). Springer: Berlin Heidelberg, 2013.
25. Kraus T, Ferreau HJ, Kayacan E, Ramon H, De Baerdemaeker J, Diehl M, Saeys W. Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles. *Computers and Electronics in Agriculture* 2013; **98**:25–33.
26. Bock HG, Plitt KJ. A multiple shooting algorithm for direct solution of optimal control problems. *Proceedings 9th IFAC World Congress Budapest*, Budapest, Hungary, 1984; 242–247. Pergamon Press.
27. Sargent RWH, Sullivan GR. The development of an efficient optimal control package. In *Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2*, Stoer J (ed.). Springer: Heidelberg, 1978; 158–168.
28. Albersmeyer J, Diehl M. The lifted Newton method and its application in optimization. *SIAM Journal on Optimization* 2010; **20**(3):1655–1684.
29. Bock HG. Recent advances in parameter identification techniques for ODE. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, Deuflhard P, Hairer E (eds). Birkhäuser: Boston, 1983; 95–121.
30. Tran-Dinh Q, Savorgnan C, Diehl M. Adjoint-based predictor-corrector sequential convex programming for parametric nonlinear optimization. *SIAM Journal on Optimization* 2012; **22**(4):1258–1284.
31. Houska B, Ferreau HJ, Diehl M. ACADO toolkit – an open source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods* 2011; **32**(3):298–312.
32. Zanon M, Horn G, Gros S, Diehl M. Control of dual-airfoil airborne wind energy systems based on nonlinear MPC and MHE. *European Control Conference*, Strasbourg, France, 2014; 1801–1806.
33. Papastavridis JG. *Analytical Mechanics*. Oxford University Press, Inc., 2002.
34. Hairer E, Nørsett SP, Wanner G. *Solving Ordinary Differential Equations I* (2nd edn), Springer Series in Computational Mathematics. Springer: Berlin, 1993.
35. Hairer E, Wanner G. *Solving Ordinary Differential Equations II – Stiff and Differential-algebraic Problems* (2nd edn). Springer: Berlin Heidelberg, 1991.
36. Griewank A, Walther A. *Evaluating Derivatives* (2nd edn). SIAM: Philadelphia, PA, 2008.
37. Albersmeyer J. Adjoint-based algorithms and numerical methods for sensitivity generation and optimization of large scale dynamic systems. *Ph.D. Thesis*, Ruprecht-Karls-Universitität Heidelberg, Baden-Württemberg, Germany, 2010.
38. Zanon M, Frasch J, Diehl M. Nonlinear moving horizon estimation for combined state and friction coefficient estimation in autonomous driving. *Proceedings of the European Control Conference*, Zurich, Switzerland, 2013; 4130–4135.
39. Serban R, Hindmarsh AC. CVODES: the sensitivity-enabled ODE solver in SUNDIALS. *Proceedings of IDETC/CIE 2005*, Long Beach, California, USA, 2005; 257–269.
40. Enright WH, Jackson KR, Nørsett SP, Thomsen PG. Interpolants for Runge-Kutta formulas. *ACM Transactions on Mathematical Software* 1986; **12**:193–218.
41. Allgöwer F, Badgwell TA, Qin JS, Rawlings JB, Wright SJ. Nonlinear predictive control and moving horizon estimation – an introductory overview. In *Advances in Control, Highlights of ECC'99*, Frank PM (ed.). Springer: London, 1999; 391–449.
42. Magni L, De Nicolao G, Magnani L, Scattolini R. A stabilizing model-based predictive control for nonlinear systems. *Automatica* 2001; **37**(9):1351–1362.
43. Nicolao GD, Magni L, Scattolini R. Stabilizing receding-horizon control of nonlinear time varying systems. *IEEE Transactions on Automatic Control* 1998; **AC-43**(7):1030–1036.
44. Diehl M, Magni L, Nicolao GD. Efficient NMPC of unstable periodic systems using approximate infinite horizon closed loop costing. *Annual Reviews in Control* 2004; **28**(1):37 –45.