acados – a modular open-source framework for fast embedded optimal control

Robin Verschueren · Gianluca Frison · Dimitris Kouzoupis · Niels van Duijkeren · Andrea Zanelli · Branimir Novoselnik · Jonathan Frey · Thivaharan Albin · Rien Quirynen · Moritz Diehl

Received: date / Accepted: date

Abstract The acados software package is a collection of solvers for fast embedded optimization, intended for fast embedded applications. Its interfaces to higher-level languages make it useful for quickly designing an optimization-based control algorithm by putting together different algorithmic components that can be readily connected and interchanged. However, since the core of acados is written in a low-level language close to the hardware, we do not sacrifice computational efficiency. Thus, we aim to provide both flexibility and efficiency through modularity, without the need to rely on automatic code generation, which facilitates maintainability and extensibility. The main features of acados are: efficient optimal control algorithms targeting embedded devices.
implemented in C, linear algebra based on the high-performance BLASFEO library, user-friendly interfaces to MATLAB and Python, and compatibility with the modeling language of CasADi. acados is published under the BSD 2-Clause open source license.

**Keywords** Direct optimal control · Optimization algorithms

**Mathematics Subject Classification (2010)** 49-04

1 Introduction

Embedded optimization, according to the definition in [18], is solving optimization problems *autonomously* and *with limited resources*. There exist many algorithms for embedded optimization and quite a few were successfully applied on real-time and embedded systems [17,39,41,64]. For an overview, we refer the reader to [18]. Embedded optimal control refers to a specific class of methods within embedded optimization: it focuses on calculating optimal decisions in order to control a dynamic system as the state changes. One of the most popular approaches in embedded optimal control is model predictive control (MPC) [42,52,31]. It is based on predicting the future behavior of a system and using this information to optimize for the action at the current time step. In linear or linear-quadratic MPC, the constraints, including the dynamic model, are affine and the objective is quadratic. Within nonlinear MPC (NMPC), some or all of the constraints and objective are nonlinear functions.

A related problem is that of moving horizon estimation (MHE) for estimating states and parameters online. Although many of the algorithms in this paper apply to MHE as well, we will not discuss it in more detail.

Historically, MPC was primarily applied to systems with long timescales, most notably in chemical processing [45], due to the fact that during each time step, a computationally costly optimization problem has to be solved. More recent algorithmic developments [12,40,44,7] and increasingly powerful embedded hardware render MPC real-time feasible for applications with shorter timescales such as autonomous driving, robotics, and avionics. A big role in bringing MPC to real-time applications is played by the implementation of efficient embedded optimal control methods, giving rise to software packages such as MPT [34] for explicit MPC, qpOASES [19], an active-set solver for quadratic programming (QP), FORCES [14,63], an interior-point solver for quadratically constrained QP (QCQP) and nonlinear programs with optimal control structure, and the ACADO Code Generation tool [35] for tailored SQP-based NMPC solvers. Other examples of nonlinear embedded optimization packages are VIATOC [36], GRAMPC [16] and FalcOpt [55].

The challenge in developing software for embedded optimal control lies in the trade-off between flexibility, memory usage and speed. Many of the software packages mentioned above are based on *automatic code generation*. They

---

1 In this paper, we focus on finite-dimensional problems, i.e., the objective is not a functional.
successfully achieve flexibility from a high-level perspective, which enables the user to design their algorithm in a high-level language like MATLAB or Python. If there is a need to run the control algorithm on embedded hardware, it can be readily translated to a more performant and embeddable language, often C or C++.

Let us focus on a particular tool, namely the ACADO Code Generation tool. One feature of ACADO code generation is the possibility to have tailored code: it is written by the software tool for one specific application only. This often incurs lower memory usage and/or lower runtimes. One disadvantage is the loss of flexibility: the code is generated for the fixed dimensions of the problem only. Any conceptual change requires regeneration and recompilation of the solver. Another disadvantage of code generation is that the resulting code is often hard to read by humans, which makes the process of debugging the embedded code challenging. Furthermore, it can be hard to predict if an automatic code optimization strategy, like loop unrolling, is beneficial in general or if it is perhaps counterproductive for some types of problems. In some cases, a compiler or a human could generate faster or more memory-efficient code. Lastly, automatic code generation, if used at all, is most effective for the most computationally expensive subroutines, typically a small number of linear algebra operations. The contribution of code generating other parts of the algorithm is less obvious and is not always justified.

For the above reasons, acados does not rely on automatic code generation. Instead, for the linear algebra operations, we make use of the recently developed high-performance linear algebra package BLASFEO [25] which outperforms code-generated triple-loop linear algebra routines and state of the art BLAS and LAPACK implementations in embedded optimization applications. Furthermore, we perform no a priori automatic code optimizations, instead we delegate that task to the optimizing compiler.

Another challenge for embedded optimal control software is related to the process of software development. Often, to not sacrifice speed of execution and/or memory footprint, embedded optimal control software uses global data and suffers from tight coupling between algorithmic components. This might lead to a codebase that is difficult to understand, maintain and extend. We choose, as opposed to other embedded optimal control software packages, to avoid these pitfalls by not unnecessarily sacrificing maintainability and readability of the codebase for a small gain in efficiency and/or a reduction of memory footprint. We try to achieve this goal by organizing the code in a modular fashion, with formal interfaces between the different algorithmic components, as described in Section 4. This allows for a straightforward way of interchanging solvers, routines, and libraries needed for the embedded control algorithm.

A final aspect of embedded optimal control software that affects flexibility, memory and runtime is the choice of modeling language and automatic differentiation tool. Several of modeling languages exist, e.g., Mathematica, sympy or the MATLAB Symbolic Toolbox. Many of these languages make use of expression trees to represent mathematical functions, which potentially leads
to a large code size, high memory usage and slow evaluation of higher-order
derivatives for non-trivial models. On the contrary, the CasADi [5] modeling
language is based on expression graphs. This often leads to shorter instruction
sequences and to smaller, typically faster code, which makes it more suitable
for embedded applications. Also, it is free and open-source software. For these
reasons, we favor CasADi for modeling nonlinear functions and differential-
algebraic equations. Additionally, acados supports the use of hand-written or
code-generated dynamic models as C source files, if they follow a certain calling
convention.

In summary, the contribution of this paper is a new software package for
embedded optimal control, called acados. It offers the following main features:

– efficient optimal control algorithms implemented in C,
– modular architecture enabling rapid prototyping,
– interfaces to Python and MATLAB,
– high-performance linear algebra based on BLASPEO [25],
– compatible with CasADi expressions [5],
– deployable on a variety of embedded devices,
– available as open-source code.

The remainder of the paper is organized as follows: in Section 2 we review
some important elements of nonlinear embedded optimization algorithms relevant to
acados. We discuss recent advances in embedded optimization algo-
rithms that motivate the development of acados in Section 3. The software
package acados itself is introduced in Section 4. Various numerical experi-
ments including hardware-in-the-loop simulations as well as comparisons to
other embedded optimization packages are presented in Section 5 and the
paper is concluded in Section 6.

2 Algorithmic ingredients for embedded nonlinear optimal control

In this paper, we consider nonlinear optimal control problems (OCP) of the form

\[
\begin{array}{c}
\text{minimize} & \int_{t_0}^{T} \ell(x(t), z(t), u(t)) \, dt + M(x(T)) \\
\text{subject to} & x(0) = x_0, \\
& 0 = f(x(t), z(t), u(t)), \quad t \in [0, T], \\
& 0 \geq g(x(t), z(t), u(t)), \quad t \in [0, T].
\end{array}
\]

In this notation, \( x : \mathbb{R} \to \mathbb{R}^{n_x} \) denotes the differential states, \( z : \mathbb{R} \to \mathbb{R}^{n_z} \) are the algebraic variables and \( u : \mathbb{R} \to \mathbb{R}^{n_u} \) denotes the control inputs. Furthermore, we use \( \ell : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \to \mathbb{R} \) for the Lagrange term or running cost, \( M : \mathbb{R}^{n_x} \to \mathbb{R} \) for the Mayer term or terminal cost, the dynamics are modeled
with a set of implicit differential-algebraic equations (DAE) with right-hand-
side \( f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \). In the remainder of the paper, we
assume the implicit DAE to be of index 1, i.e. $\frac{\partial f}{\partial \dot{x}, \partial z}$ is invertible. The nonlinear path constraints are given by $g : \mathbb{R}^{nx} \times \mathbb{R}^{nz} \times \mathbb{R}^{nu} \to \mathbb{R}^{ng}$, and the initial value of the states is $x_0 \in \mathbb{R}^{nx}$. We consider $T$, the horizon length, to be fixed.

2.1 Multiple shooting.

In acados, we discretize nonlinear OCPs with a multiple shooting approach. We introduce a time grid $[t_0, t_1, \ldots, t_N]$ with $t_k \leq t < t_{k+1}, k = 0, \ldots, N-1$. We introduce discretized state variables $x_0, x_1, \ldots, x_N$, algebraic variables $z_0, z_1, \ldots, z_{N-1}$ and controls $u_0, u_1, \ldots, u_{N-1}$. For the control trajectory, we choose a piecewise constant control parametrization. On each time interval $t_k \leq t < t_{k+1}$, we can then write the numerical simulation routine as

$$
\begin{bmatrix}
x_{k+1} \\
z_k
\end{bmatrix} = \phi_k(x_k, u_k), \quad k = 0, 1, \ldots, N - 1,
$$

where the separate components will be denoted by $\phi^x_k(x_k, u_k)$ and $\phi^z_k(x_k, u_k)$.

The multiple shooting approach can often lead to better convergence behavior compared to single shooting, where the simulation and optimization is performed sequentially, as shown in [8]. In addition to the corresponding variables in OCP (1), we introduce slack variables $s_k, k = 0, \ldots, N$ to allow for a formulation with soft constraints and $l_1, l_\infty$ penalties, among other uses of slacks. The resulting nonlinear programming (NLP) problem looks as follows:

\begin{align*}
\text{minimize} & \quad \sum_{k=0}^{N-1} l_d(x_k, s_k, z_k, u_k) + M(x_N, s_N) \\
\text{subject to} & \quad x_0 = x_0, \\
& \quad \begin{bmatrix}
x_{k+1} \\
z_k
\end{bmatrix} = \phi_k(x_k, u_k), \quad k = 0, 1, \ldots, N - 1, \\
& \quad 0 \geq g_k(x_k, z_k, u_k) + s_k \quad k = 0, 1, \ldots, N - 1, \\
& \quad 0 \geq g_N(x_N) + s_N, \\
& \quad 0 \leq s_k \quad k = 0, 1, \ldots, N.
\end{align*}

The above NLP could be solved by any general-purpose NLP solver, like IPOPT [51]. The current scope of acados, however, encompasses efficient embedded optimal control methods for solving such structured NLPs, since these are better suited in a real-time and/or embedded setting [13]. Sequential Quadratic Programming (SQP) is an example of an embeddable optimization method.
2.2 Sequential Quadratic Programming and Real-Time Iterations

We briefly present the structure of an SQP algorithm as implemented in acados. At the very least, an embedded SQP algorithm should feature:

- numerical integration of the continuous-time dynamics,
- generation of first-order and possibly second-order sensitivities of objective and constraints,
- a procedure for approximating the Hessian matrix,
- an efficient QP solver (typically developed separately)
- initialization strategies for warm-starting the algorithm for the next problem.

Note that globalization strategies such as line search or trust regions are considered out of scope for this paper, given the stringent real-time demands for embedded optimization. In summary, the SQP algorithm in acados looks as follows:

\[ w_r^{i+1} = w_r^i + \Delta w_{QP}, \quad i = 0, 1, \ldots, \]
\[ \pi_r^{i+1} = \pi_{QP}, \quad i = 0, 1, \ldots, \]
\[ \mu_r^{i+1} = \mu_{QP}, \quad i = 0, 1, \ldots, \]

where \( w_r^i = [x_0^i, \ldots, u_0^i, \ldots, x_N^i] \) is the primal iterate at SQP iteration \( i \), \( \lambda^i \) and \( \mu^i \) are the dual iterates, readily available from the QP solution. Note that the algebraic variables have been eliminated from the OCP, but numerical approximations of these values are accessible from the numerical integration routine. The step \( \Delta w_{QP} \) is computed by solving following QP:

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=0}^{N-1} \left[ \Delta x_k \right]^T \begin{bmatrix} H_k & \Delta x_k^T \\ \Delta u_k & \Delta u_k \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + \Delta x_N^T Q_N \Delta x_N + q_N^T \Delta x_N \\
& \quad + \sum_{k=0}^{N} s_k^T P_k s_k + p_k^T s_k \\
\text{subject to} & \quad \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + \phi_k^x - x_{k+1}, \quad k = 0, \ldots, N-1, \\
& \quad \Delta x_0 = x_0, \\
& \quad -g_k \geq G_k^x \Delta x_k + G_k^u \Delta u_k + s_k, \quad k = 0, \ldots, N-1, \\
& \quad -g_N \geq G_N^x \Delta x_N + s_N, \\
& \quad 0 \leq s_k \quad k = 0, \ldots, N.
\end{align*}
\]

(3)
Above, we used these shorthands: \( \phi^{k}(x_k, u_k) \), \( g(x_k, z_k, u_k) \) and \( g_N(x_N) \) are denoted by \( \phi_k^{k}, g_k, k = 0, 1, \ldots, N - 1 \) and \( g_N \), respectively. We formulate the NLP in such a way that the slack variables appear directly as tailored optimization variables. We note that some, but not all QP solvers can deal with slack variables directly. For those that do not, slack variables are reformulated as extra input variables.

In the remainder of this section, we discuss each of the ingredients of an efficient SQP solver, described above, separately. Generation of sensitivities using numerical integration will be treated in Section 2.3, Hessian approximation in Section 2.4, structure-exploiting QP solvers in Section 2.5 and real-time considerations in Section 2.6.

2.3 Numerical simulation and sensitivities

An important part of the implementation of direct shooting methods for optimal control consists of reliably and efficiently computing numerical simulation and sensitivity results for the nonlinear system of differential-algebraic equations that represents a dynamic model for our particular system of interest.

Within the family of single-step integration methods one typically distinguishes between explicit and implicit schemes \([32]\). Well-known examples of explicit integration schemes include explicit Runge-Kutta (RK) formulas such as explicit Euler and the RK method of order 4. Explicit integration schemes are easy to implement since they rely on a direct combination of explicit evaluations of the right-hand side of the system dynamics. Instead, implicit integration schemes result in a nonlinear system of equations that implicitly defines the numerical simulation result. Unlike explicit integration methods, the nonlinear system in implicit integration schemes generally needs to be solved numerically using an iterative procedure such as a Newton-type method. However, implicit formulas are very popular in practice because of their improved numerical stability properties and higher order of accuracy. Especially in case of stiff dynamical systems and implicit or differential-algebraic equations, an implicit integration scheme should often be used \([33]\).

When using these numerical integration schemes within direct multiple shooting, one additionally needs a computationally efficient and reliable way of computing first (and possibly second) order derivatives of the simulation results with respect to the state and control input values:

\[
\frac{\partial \phi(x_k, u_k)}{\partial x_k}, \quad \frac{\partial \phi(x_k, u_k)}{\partial u_k}, \quad \sum_{i=1}^{\pi_x} \pi_{k,i} \frac{\partial^2 \phi(x_k, u_k)}{\partial^2 (x_k, u_k)},
\]

where \( \pi_k \in \mathbb{R}^{\pi_x} \) is called the seed vector, for which the Lagrange multipliers are used to compute the exact Hessian of the Lagrangian. As discussed in more detail by \([40]\), sensitivity propagation for direct optimal control methods is typically based on a discretize-then-differentiate type of approach such as internal numerical differentiation (IND) in \([9]\). For the class of explicit integration methods, this concept leads to a forward or backward sensitivity
propagation based on algorithmic differentiation (AD) techniques [29]. In case of an implicit integration scheme, the IND approach either results in iterative differentiation techniques or a direct computation of sensitivities based on the implicit function theorem [2]. In addition, forward-backward propagation schemes can be derived to compute the symmetric Hessian contributions [50].

Recent work in [49,48] proposed an algorithmic approach to embed implicit integration schemes with sensitivity analysis in Newton-type optimization for direct optimal control without the need for any iterative procedure, based on the concepts of numerical condensing and expansion in a lifted Newton-type optimization method [3].

2.4 Convex Hessian Approximation Methods

**Gauss-Newton Hessian approximation.** In the case of a (nonlinear) least-squares objective in (2), e.g. when tracking a reference, we have

\[
\begin{align*}
  l(x_k, u_k) &= \|r(x_k, u_k)\|_2^2, \quad k = 0, \ldots, N - 1 \\
  M(x_N) &= \|r_N(x_N)\|_2^2,
\end{align*}
\]

with \( r : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_r}, r_N : \mathbb{R}^{n_x} \to \mathbb{R}^{n_r} \). Notice that this kind of residual function is a common case in embedded optimization.

The Gauss-Newton Hessian approximation amounts to

\[
H_{GN} = \text{blkdiag}(H_{GN}^0, \ldots, H_{GN}^N)
\]

at the linearization point \( w[i]^\top = [x_0[i]^\top, u_0[i]^\top, \ldots, x_N[i]^\top] \), where the Hessian blocks are defined as follows:

\[
\begin{align*}
  H_{GN}^k &= \left( \frac{\partial r}{\partial x,u}(x_k^i, u_k^i) \right)^\top \left( \frac{\partial r}{\partial x,u}(x_k^i, u_k^i) \right), \quad k = 0, \ldots, N - 1, \\
  H_{GN}^N &= \left( \frac{\partial r_N}{\partial x}(x_N^i) \right)^\top \left( \frac{\partial r_N}{\partial x}(x_N^i) \right).
\end{align*}
\]

Since no second-order sensitivities are necessary, the Gauss-Newton Hessian approximation offers a competitive alternative to SQP with exact Hessians, although it converges linearly only. We remark that for a quadratic objective function in (2), the same quadratic objective arises in (3), and no additional computations are needed. For more details on Gauss-Newton methods in the context of NMPC, we refer the reader to [30].

**Sequential Convex Quadratic Programming (SCQP).** A generalization to using SQP with a Gauss-Newton Hessian approximation is sequential convex quadratic programming [58]. In a sense, a Gauss-Newton SQP algorithm neglects any curvature present in the inequality constraints by linearizing them. In practice however, convex-over-nonlinear objectives and/or constraints of the form

\[
g(x,u) = \varphi(c(x,u)),
\]
with \( \varphi \) a convex function arise often in practice. Examples include ellipsoidal terminal constraints to ensure stability of an NMPC scheme, the friction ellipse in automotive applications, or tube-following for robotic manipulators.

In SCQP, we still linearize the inequalities, but bring the convex contributions from the inequality constraints, multiplied with a Lagrange multiplier, into the Hessian approximation. For example, consider the following OCP:

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=0}^{N-1} \psi_k(r_k(x_k, u_k)) + \psi_N(r_N(x_N)) \\
\text{subject to} & \quad x_0 = \mathbf{x}_0, \\
& \quad x_{k+1} = \phi^i(x_k, u_k), \quad k = 0, 1, \ldots, N - 1, \\
& \quad 0 \geq \varphi_k(c_k(x_k, u_k)) \quad k = 0, 1, \ldots, N - 1, \\
& \quad 0 \geq \varphi_N(c_N(x_N)).
\end{align*}
\]

with \( \psi_k, \psi_N, \varphi_k \) and \( \varphi_N \) convex functions. The resulting SCQP Hessian approximation at \( w^{[i]} \) reads

\[
H_k^{\text{SCQP}} = D_k^T \nabla_r^2 \psi_k \left( r_k(x_k, u_k) \right) D_k + \sum_{i=0}^{n_n} \lambda_i C_k^T \nabla_c^2 \varphi \left( c_k(x_k, u_k) \right) C_k, \\
H_N^{\text{SCQP}} = D_N^T \nabla_r^2 \psi_N \left( r_N(x_N) \right) D_N + \sum_{i=0}^{n_n} \lambda_i C_N^T \nabla_c^2 \varphi \left( c_N(x_N) \right) C_N,
\]

where we define

\[
D_k = \left( \frac{\partial r_k}{\partial (x, u)} \left( x_k, u_k \right) \right) \quad C_k = \left( \frac{\partial c_k}{\partial (x, u)} \left( x_k, u_k \right) \right) \\
D_N = \left( \frac{\partial r_N}{\partial x} \left( x \right) \right) \quad C_N = \left( \frac{\partial c_N}{\partial x} \left( x \right) \right).
\]

By inspection, we see that the SCQP Hessian is always positive semi-definite. For problems that feature convex-over-nonlinear constraints, this Hessian contribution offers better convergence guarantees than a Gauss-Newton Hessian [58]. For more details on the implementation details of SCQP in acados, see [59].

**Structure-preserving convexification with minimal regularization.** In the last two paragraphs, we devised two Hessian approximations which are convex by construction. However, when the exact Hessian of the Lagrangian is used, it might be indefinite. When this happens, the optimal direction \( \Delta w_{QP} \) cannot be guaranteed to be a descent direction. Furthermore, many QP solver codes expect a positive (semi-)definite Hessian, even if the second order sufficient conditions for optimality are met. The aim of regularization is to obtain an approximation \( \tilde{H} = \text{blkdiag}(\tilde{H}_0, \ldots, \tilde{H}_N) \) with each \( \tilde{H}_k > 0 \). We very briefly
discuss three different methods here and compare their convergence in an SQP-type setting in Section 5.

Let \( V_kD_kV_k^{-1} \) be the eigenvalue decomposition of \( H_k \), for \( k = 0, \ldots, N \). Two simple ways of regularizing the Hessian blocks are

\[
\tilde{H}_k = \text{project}(H_k, \epsilon) := V_k \left[ \max(\epsilon, D_k) \right] V_k^{-1}, \quad (6a)
\]

\[
\tilde{H}_k = \text{mirror}(H_k, \epsilon) := V_k \left[ \max(\epsilon, \text{abs}(D_k)) \right] V_k^{-1}, \quad (6b)
\]

with \( \epsilon \) small and \( \max(\cdot), \text{abs}(\cdot) \) defined elementwise. A better convergence behavior can be reached by exploiting the optimal control problem structure of (3). Such an approach has been proposed in [60]. The difference with more naive regularization methods as stated above, is that it first exploits as much convexity as possible through the optimal control structure, before regularizing the remaining negative directions. The result of the approach in [60], which we call \textit{convexification}, is a positive definite Hessian approximation \( \tilde{H} \). The complexity of the regularization method is linear in the horizon length, and under some conditions, the SQP iterates converge quadratically to a local solution. An efficient implementation of this new Hessian regularization method is included in \textit{acados}.

2.5 Structure-exploiting QP solvers

There exist different solution strategies for QP (3), we briefly describe three of them in this section. We note that linear-quadratic optimal control problems can be efficiently solved with the same kind of QP solving strategies presented. As such, \textit{acados}, conceived as a modular software package, can also be used to facilitate solving linear-quadratic QPs, as in linear-quadratic MPC.

**Sparse approach.** OCP (3) can be solved directly by using a general-purpose sparse QP solver, e.g., HQP [20], OOQP [28] or OSQP [54].

**Structured approach.** OCP (3) is solved by exploiting its multi-stage structure, but dense linear algebra is used. An example is the approach from [51] and in solvers like FORCES [14,63], HPMPC [27] and HPIPM [1].

**Condensing approach.** An alternative to the previous approaches is the so-called \textit{condensing} approach [11]. By eliminating dynamic equality constraints in (3), we obtain a smaller QP with only the control inputs and possibly the initial state as optimization variables. Any general-purpose dense QP solver can then be used to solve the smaller QP, e.g. \textit{qpOASES} [19]. Condensing can be shown to be of quadratic complexity in the horizon length [23].
Partial condensing. A mix between the two previous approaches (i.e. structured, condensing) for solving (3) can be obtained by not eliminating all state variables, but only per blocks of \( N/N_2 \) stages (we assume for simplicity that \( N \) is an integer multiple of \( N_2 \), where \( N_2 \) is the ‘new’ horizon length of the partially condensed problem. By this additional degree of freedom, partial condensing enables us to find a better trade-off between horizon length and number of optimization variables, for a given problem. For more details on partial condensing, the reader is referred to [6].

So far, we disregarded any real-time and embedded requirements. In an embedded optimization setting, such as model predictive control (MPC), solution strategies become more challenging since problems need to be solved online in a dynamically changing environment. Next, we treat QP strategies tailored to these needs.

2.5.1 Embedded QP solvers

In an embedded (i.e. online) context for optimal control, it is important to realize that the problem in (3) depends parametrically on the initial state \( x_0 \), for which the values at subsequent time steps lie close to each other. Embedded optimization codes often exploit this fact, either by storing results of computations made in the past, or by proactively computing some quantities needed at a later point in time. We now discuss a number of solution strategies, interfaced from acados.

First-order methods. A number of first-order methods have been proposed for solving problem (3), mainly based on either the fast gradient method or the alternating direction method of multipliers (ADMM). The strict real-time requirements for solution methods make first-order methods a viable candidate. However, they might suffer from slow convergence rates. For an overview of first-order methods in the context of embedded optimal control, see [18,37]. An example of a first-order embedded QP solver is OSQP [54].

Active-set methods. Active-set methods are based on the principle that if one happens to know the set of active constraints at the solution, the optimization reduces to the solution of a single set of linear equations, which is substantively simpler than solving a general QP. Thus, active set methods propose in each iteration a current guess of the set of active constraints (also called the working set), and, when not at the solution, update this guess accordingly. For the QP solver qpOASES [19], active set updates are based on the fact that subsequent problems in real-time MPC lie close to each other. qpOASES is best suited for small and dense problems, and it is typically used in combination with the condensing approach. The fundamental idea behind qpOASES is based on a parametric homotopy technique for the initial state dependency. Since qpOASES is able to reuse information (i.e. warm-starting) from one problem to the next, it is particularly well-suited for (N)MPC.
Interior-point methods. The fundamental idea behind interior-point methods is to use a Newton-type scheme to solve a set of nonlinear equations, namely the KKT conditions corresponding to optimization problem (3), perturbed with a decreasing barrier parameter. An important ingredient of interior-point methods is how to update the barrier parameter from one iteration to the next. The Mehrotra predictor-corrector scheme [43] is by far the most popular one in interior-point methods for quadratic programming. In [53,51], a structure-exploiting strategy for linear MPC problems is presented, which inspired others to write efficient interior-point solvers. An example of structure-exploiting interior-point solvers based on a Riccati recursion for the linear system solution is HPMPC [27], and its successor HPIPM [1]. They use a similar strategy as presented in [51], but additionally offer linear algebra kernels optimized for small matrices, optimized for different computer architectures, which enabled an additional computational speedup compared to existing methods. Additionally, they provide an efficient implementation of full and partial condensing, which are embedded in the solver for ease of use.

Dual-Newton strategy. The dual Newton strategy is an algorithm that is based on dual decomposition tailored to linear-quadratic OCPs in the form of (3), with an open-source implementation in the software qpDUNES [21].

A main advantage of the dual Newton strategy, as in most active set methods, is warm-starting. Contrary to qpOASES, qpDUNES can perform multiple active set changes per iteration. However, a premature termination of the algorithm does not return a meaningful solution as in the case of qpOASES since it is not feasible nor optimal for a neighboring problem. As recently observed in [38], the convergence of qpDUNES can benefit significantly from a partial condensing preprocessing step.

2.6 Real-time iterations

In a real-time control setting, we solve NLP (2) in sequence and under stringent time conditions. Since the environment is anyway changing continuously, it is often sufficient to solve it approximately – it is of no use of having a high-accuracy but past-the-deadline solution.

One such online method is the real-time iteration (RTI) scheme [12]. It solves an inequality-constrained QP in each iteration. The resulting generalized predictor is better suited for predictions across active set changes, than e.g. a tangential predictor obtained from an interior-point method. For a brief overview, we refer the reader to [13].

In each RTI, one full iteration of an SQP-type scheme is performed, including generation of the sensitivities w.r.t. all variables. We could introduce additional approximations by not updating all sensitivities in each RTI [65]. Such approximations exist on different levels: from only updating the initial state constraint with the current estimate of the state of the system, over updating the right-hand side of the (in)equality constraints, to the full RTI. By
interleaving different approximations at different sample times, we obtain a multi-level iterations scheme, as introduced by [10].

3 Algorithm implementations in acados

In this section, we focus on the algorithm implementations in acados. Since acados builds on other software to handle basic linear algebra operations (BLASFEO [25]) and QPs (HPIPM [1]), these will be presented first. Afterwards, a short description of integrators and SQP-type optimization solvers will be given.

3.1 Linear algebra: BLASFEO

At the heart of all embedded optimization routines lies either an implementation of a small set of linear algebra routines (e.g. matrix-matrix multiplication, Cholesky decomposition), or a call to a specialized linear algebra library (e.g. BLAS and LAPACK). Generally BLAS implementations focus on performance for large dense matrices, as used in high-performance computing and data science. Considerably less investigation exists in BLAS and LAPACK implementations for small dense matrices.

Often, the linear algebra code in embedded optimization packages, for example in the ACADO Code Generation tool or Forces Pro, is code-generated. For very small matrix sizes (e.g. 4 × 4), this technique outperforms optimized linear algebra libraries. Furthermore, code-generation has the advantage that the code can be kept ‘library-free’. However, for larger matrix sizes (e.g. in the range 10 × 10 to 100 × 100, typical in MPC applications), code-generated linear algebra routines underperform with respect to optimized libraries.

BLASFEO [25] is a linear algebra package that targets computations for small matrices. It offers highly optimized linear algebra routines (e.g. dSYRK, Dgemm, dpotrf), tailored for the matrix sizes typically encountered in embedded optimization. These routines exploit architecture-specific vector instructions for floating point operations (e.g. AVX), and focus on performance for matrices fitting in cache.

Furthermore, BLASFEO defines a packed matrix format (called panel major) which optimizes the cache usage, guaranteeing close to peak performance for matrices of sizes up to a couple hundreds. All high-performance BLASFEO routines use this panel major matrix format, and there is a rich set of auxiliary routines to operate on this matrix format, as well as to convert from/to column- or row-major formats. In this sense, BLASFEO provides a complete linear algebra framework, which can be used to implement many fast optimization algorithms.

Except for trivially small matrices, BLASFEO enables a considerable speedup (10× for some matrix sizes) in the matrix computations, compared to code-generated linear algebra kernels. For all small matrix sizes up to, say, 300 ×
300, BLASFEO offers a considerable speedup compared to state-of-the-art BLAS implementations, like OpenBLAS, too. The use of BLASFEO is one of the factors why acados performs better than ACADO on medium-scale problems, as we will see in Section 5.

3.2 Quadratic programming: HPIPM

In the implementation of SQP-type algorithms, QP sub-problems need to be solved efficiently at each iteration. The QP sub-problem solution is typically one of the two most expensive steps in SQP schemes, the other being the simulation and sensitivity computations of dynamical systems.

HPIPM [1] is a library defining three QP types (dense QP, OCP QP and tree-structured OCP QP, all supporting soft constraints), and a rich set of routines to create, manage and solve the QPs. All QP solvers are Mehrotra’s type primal-dual interior point methods, and they are implemented using the BLASFEO linear algebra framework.

Furthermore, HPIPM provides a set of routines to convert between the different QP types. Condensing routines convert an OCP QP into a dense QP; expansion routines convert a dense QP solution into an OCP QP solution. Partial condensing routines convert an OCP QP into another OCP QP with shorter horizon length; partial expansion routines convert an OCP QP solution into another OCP QP solution with longer horizon length. An overview of the algorithms implemented in HPIPM is found in [24].

In acados, the QP framework is based on HPIPM, in the sense that HPIPM provides both the dense QP and OCP QP definitions, as well as (partial) condensing algorithms to convert them and interior-point methods to solve them. Numerous other QP solvers are then interfaced to acados to alternatively solve the same types of QP problems. At the time of writing, interfaces exist to OOQP, qpDUNES, qpOASES, and OSQP.

3.3 Numerical simulation

acados features different kinds of numerical simulation routines. There are implementations of explicit and implicit Runge-Kutta integrators available, both of which support the optional propagation of first-order forward and adjoint sensitivities, as well as second-order sensitivities. The explicit integrators can be used with explicit ODE models and supports different Butcher tableaus, including Euler’s method and RK4. Moreover, the implicit integrators can be used with an index-1 differential-algebraic equation (DAE) or implicit ODE model and use the Gauss-Legendre Butcher tableaus. A novel implementation for lifted collocation integrators [49] has been made part of acados, as well as a recently proposed structure-exploiting IRK algorithm, the so-called GNSF-IRK scheme [22], discussed next.
The concept of GNSF-IRK is to rigorously exploit the linear dependencies within the dynamic system. It extends the ideas of the linear input and linear output subsystems that have been implemented within the ACADO Code Generation tool \cite{47} and uses a more flexible structured dynamic system formulation that can also handle index-1 DAEs. A main challenge for structure-exploiting integrators is to appropriately reformulate the dynamic system of interest into the desired structured form. \texttt{acados} features an automatic transcription method for the GNSF structure \cite{22}, implemented as a MATLAB function for CasADi models.

A last important feature of \texttt{acados} is that integrators can vary from stage to stage, with e.g. different state and control dimensions, different integration step length, or different integration schemes.

We note that all integrator modules, except for the ERK integrator, are based on hardware tailored linear algebra routines in BLASFEO to speed up the LU factorizations and the corresponding triangular system solutions, as discussed in \cite{26}.

3.4 SQP-type methods

For nonlinear programming, \texttt{acados} offers different SQP-like methods. A full-step SQP method is available, with different algorithmic options. As Hessian approximations, we have Gauss-Newton Hessians, SCQP, and exact Hessians with regularization/convexification as discussed in Section \cite{26} and we allow for user-defined Hessian approximations. For use in an online setting, e.g. in NMPC, a specialized RTI routine is available.

Both the SCQP algorithm and the convexification method of Section \cite{24} are novel features, to the authors’ knowledge, not present in any other NMPC software packages.

4 The \texttt{acados} software package

\texttt{acados} implements some of the optimization methods mentioned in the previous sections. It is meant to be user-friendly at a high level, and efficient at a low level. In order to balance these properties, we developed a core library written in C which exposes functionality to the Python and MATLAB interfaces. In this section, we first discuss the functionality of this inner core module, we then describe internal and external interfaces that are crucial for usability.

4.1 The \texttt{acados} core library

The embedded optimization algorithms discussed in Section \cite{2} are implemented in \texttt{acados} in a modular fashion. For example, there is a clear contract for the interface between an NLP solver and an integrator. The integrator expects a
Table 1 Overview of the software modules present in acados.

| Module       | Variants                               |
|--------------|----------------------------------------|
| OCP QP       | HPiPM, qpDUNES, HPMPC, DQQP, OSSQP     |
| Dense QP     | HPiPM, qpOASES, DQQP                   |
| Condensing   | Full condensing (HPiPM)Partial condensing (HPiPM) |
| Simulation   | ERK, IRK, GNSF-IRK, lifted IRK         |
| OCP NLP      | Gauss-Newton SQP, Gauss-Newton SCQP, Exact-Hessian SQP, RTI |
| Regularization | Projection, Mirroring, Convexification |
| Nonlinear function | CasADi generated functions, C-code functions |

linearization point \( w^{[l]} \) and returns the end state of a simulated trajectory, and optionally first- and second-order sensitivities:

\[
\text{NLP solver} \xrightarrow{\text{lin. point}} \text{integrator} \xrightarrow{\text{sim., sens.}}
\]

Similar diagrams can be drawn for all other algorithmic components, including (partial) condensing, QP solvers, function evaluations etc. Each of these algorithmic components are modeled within acados as separate modules. Some modules can be used as standalone modules, or in combination with others. For instance, depending on the choice of algorithm, an NLP solver will make use of some or all of the other modules. In Table 1 we see an overview of all modules currently present in acados, together with the implemented algorithmic variants.

It is an important design choice that all modules are identical in their signature. That way, all modules look similar to the users of acados. For developers, it should be straightforward to extend acados with another module. The signature is as follows (in C syntax):

```c
int <solver>(void *config,
            void *dims,
            <module>_in *in,
            <module>_out *out,
            void *opts,
```
Here, `<module>` stands for the name of the module at hand, for example `ocp_qp` for QP problems with optimal control structure or `sim` for integration problems, and `<solver>` is a placeholder for a function implementing the specific solver for problems corresponding to this module, e.g., `ocp_qp_hpipm` (interface to HPIPM solver) or `sim_erk` (explicit Runge-Kutta method), etc. Each module returns an `int` which denotes a solver-specific error status – zero means successful completion in this context. All of the input arguments are pointers. Each of the arguments comes with a set of helper functions, called `...calculate_size`, computing the size (in bytes) of the `struct` pointed to, as well as a set of functions, called `...assign`, to initialize a block of memory. The usual workflow is as follows:

```c
int num_bytes = <module>_solver_config_calculate_size();
void *mem_ptr = malloc(num_bytes);
void *<module>_config = <module>_solver_config_assign(mem_ptr);
```

By making this design choice, all memory allocations happen before execution of any of the `acados` solvers and no dynamic memory allocation is needed while running the solver. This avoid the overhead of dynamic memory allocation and eliminates the risk of leaking memory, as the user of (or the interface to) the core library is fully responsible for allocating and deallocating the memory used by `acados`.

We will now look into the different arguments in a bit more detail.

`config` - a pointer to `struct` containing function pointers to the above-mentioned helper functions `...calculate_size` and `...assign`, among others.

`dims` - a pointer to `struct` with the data pertaining to the dimensions of the problem.

`in` - a pointer to `struct` with the input data to the specific solver. For a given module, this `struct` is identical for all variants.

`out` - a pointer to `struct` with the solution data from the solver. Identical for all solvers for each module.

`opts` - a void pointer that can be cast to a pointer to `struct` containing algorithmic options needed by the solver, and thus solver-specific.

`mem` - a void pointer that can be cast to a pointer to `struct` with additional memory used by the solver. This memory is to be preserved between calls to the solver.

`work` - a void pointer that points to a block of memory that functions as ‘scrap’ space, i.e., this memory does not have to be preserved between calls to the solver.

Note that the layout of the `mem` and `work` elements may depend on which solver options the user chooses.
Some modules comprise other modules. For example, an SQP solver for optimal control problems might need an integrator, which is on its own a proper acados module. In this context, we call the integrator a submodule. Each of the arguments above, dims, in, etc., have fields corresponding to submodules. For an NLP solver, the relation between it and the submodules is depicted in Figure 1. We remark that the calculation of the memory size of a module with submodules is done recursively, i.e., calling the calculate_size function on the top module returns the required memory size of the top module and all of its submodules, and submodules of submodules, etc. This allows users to allocate all the memory outside of acados, by design.

The core library of acados contains mostly what has been described in this section: a collection of modules, each with corresponding data types and variants of solvers, as well as helper functions for memory management. Using the core library directly can be cumbersome and error-prone, as many details need to be taken into account: it is designed to be efficient and flexible. To cater to the needs of the end user, we offer high-level interfaces, which are described next. The connection between the core library and its interfaces is depicted in Figure 2.

4.2 The C interface

The C interface is responsible for encapsulating the low-level constructs of the acados core.

Choosing Solvers. When working with the core library, all functions are specific to one variant of a module: when solving a QP with, say, qpOASES, the code will refer to structs like dense_qp_qpoases_memory, dense_qp_qpoases_opts, etc. We want to make abstraction of this to facilitate switching solvers easily. To this end, for each module we define a ‘plan’. A plan is a struct that contains a number of fields representing the choice of a particular combination of

![Fig. 1 Example of the relation between modules and submodules in acados for a specific case of a possible SQP algorithm.](image)
solvers. For example, the plan for an SQP-type method with Gauss-Newton Hessian approximation, for a problem discretized with an ERK integrator using HPIPM as an underlying QP solver, reads as

```c
ocp_nlp_solver_plan plan = {
    {PARTIAL_CONDENSING_HPIPM},
    {ERK, ERK, ERK, ...},
    SQP_ON,
    {LINEAR_LS, ...},
    {CONTINUOUS_MODEL, ...},
    {BGH, ...},
};
```

Here, the arrays should be of the correct length (omitted for brevity, with a slight abuse of notation). As a general rule, solvers that make use of other modules should include them in their plan.

**Passing options.** Ideally, options are passed as associative arrays (sometimes called dictionaries). Unfortunately, in C we do not have those at our disposal. For this reason, we manipulate a specific options struct with functions taking a textual representation of the option via a string. The string encodes both the module that the option belongs to, as well as the name of the option.
Memory management. Allocating memory ‘manually’ as described above can quickly become cumbersome. For this reason, we make available a few routines that automate that process. To this end, in the C interface each module from the core library is mirrored by an additional function with signature

```c
<module>_solve(<module>_solver *solver,
    <module>_in *in,
    <module>_out *out);
```

`solver` is a pointer to a C structure that encapsulates the data needed other than input and output. By doing so, we reduce the amount of boilerplate code. Pseudocode for a typical workflow could look as follows:

```c
<module>_config *config = <module>_config_create(plan);
<module>_dims *dims = <module>_dims_create(N);
<module>_opts *opts = <module>_opts_create();
// ...
// setup options in opts
// ...
<module>_solver *solver = <module>_create(config, dims, opts);
ext int status = <module>_solve(solver, in, out);
```

Convenience routines The C interface additionally offers helper routines, so-called ‘setters’ and ‘getters’, that wrap the handling of the low-level structs of the acados core.

4.3 High-level interfaces

For non-expert users of embedded NMPC software, writing C code manually can be error-prone and tedious. Therefore, we offer interfaces to three popular languages for scientific computing: C++, Python and MATLAB. As such, we created a small domain-specific language within each of these frameworks. As is depicted in Figure 2, we build on top of code from the C interface of acados. The C++, Python and MATLAB interfaces of acados use CasADi as a modeling language, which is itself a C++ code. As such, it is readily interfaced with acados: the acados high-level interfaces make use of the respective CasADi interfaces, which allows for convenient modeling and code generation of non-linear functions and derivatives. An additional benefit of using CasADi is that the solution behavior of acados can be easily compared with the solutions coming from the numerous optimization tools interfaced with CasADi.
Once the OCP to be solved is described through the domain-specific language implemented by the high-level interfaces, a human readable self-contained C project that makes use of templated code can be generated. The generated project contains all the C code necessary for function evaluations generated through CasADi and the C code necessary to set up the OCP and the NLP solver using the acados C interface. Moreover, a MATLAB S-Function and a build system that can be used to compile the code is generated.

With this workflow, it possible to obtain a self-contained, high-performance solver that can be easily deployed on embedded hardware starting from a description of the OCP in a high-level language.

We remark that model equations and other nonlinear functions are called from acados in a completely language-agnostic way: acados is at no point aware of which modeling tool is being used. One benefit is that this facilitates self-written models (in C/C++), which are also completely compatible with acados. However, this is more involved, since in the case of CasADi functions, memory allocation and matrix format conversions are taken into account automatically by in the CasADi functions wrapper in acados.

5 Numerical Results

This section consists of a few numerical experiments with acados and comparisons to other embedded optimization software packages. We discuss performance on the nonlinear chain-of-masses problem, and show one closed-loop engine control experiment on an embedded platform.

5.1 Case Study 1: Chain of Masses

As a benchmarking problem, we take the chain-of-masses problem as presented in [62]. It is useful in the sense that the problem is simple enough to understand intuitively, yet complicated enough to get non-trivial results from a range of different solvers. Also, by increasing the number of masses, one could compare behavior for different numbers of states easily, without changing much code.

5.1.1 System description

The control objective in this example is to stabilize the motion of a chain of \( M = 5 \) balls with mass \( m \) connected by springs to an equilibrium position. The mass on one end of the chain is fixed at \((0, 0, 0)\). The mass on the other end can be freely moved.

Let \( p_i \) be the position of mass \( i \), for \( i = 1, \ldots, M \). The model equations can then be derived as follows. From Hooke’s law, we know that (see Figure 3)

\[
F_{i,i+1} = D \left( 1 - \frac{L}{\|p_{i+1} - p_i\|} \right) (p_{i+1} - p_i),
\]
with each spring having spring constant $D$ and rest length $L$.

This allows us to write the equations of motion for the middle balls, which read as

$$\dot{p}_i = \frac{1}{m_i} \left( F_{i+1,i} - F_{i-1,i} \right) + g_z, \quad i = 2, \ldots, M - 1,$$

with $g_z$ the gravitational acceleration vector. For the free ball, we assume direct control of the velocity:

$$\dot{p}_M = u,$$

with $u \in \mathbb{R}^3$.

We now introduce a state space formulation with states

$$x^T = [p_2^T, p_3^T, \ldots, p_{M-1}^T, p_M^T, v_2^T, v_3^T, \ldots, v_{M-1}^T]$$

where $x \in \mathbb{R}^{n_x}$ with $n_x = 3 \cdot (2 \cdot (M - 2) + 1)$, which results in the following set of ODEs:

$$\dot{x} = f(x, u) = \begin{bmatrix} v_2 \\ \vdots \\ v_{M-1} \\ u \\ \frac{1}{m_2} (F_{2,3} - F_{1,2}) + g_z \\ \vdots \\ \frac{1}{m_M} (F_{M-1,M} - F_{M-2,M-1}) + g_z \end{bmatrix}.$$  \hspace{1cm} (7)

We remark that the only nonlinearity is introduced in the calculation of the forces. The steady state $(x_{ss}, u_{ss})$ of the system can be found by setting $f(x_{ss}, u_{ss}) = 0$ for any given $p_{M,ss}$. In all the following, we take $p_{M,ss} = [7.5, 0, 0]^T$.

### 5.1.2 Optimal control problem formulation

In order to stabilize the motion of the chain of masses to the steady state, we propose the following optimal control problem, obtained by performing a
Table 2  Design parameters for the chain of masses case study

| Quantity  | Description                        | Value                  |
|-----------|------------------------------------|------------------------|
| m         | mass of one ball                   | 0.1125 kg              |
| D         | spring constant                    | 0.4 N/m                |
| L         | rest length of the springs         | 0.1375 m               |
| g_z       | gravitational acceleration vector   | \([0, 0, -9.81\)^T\) m/s^2 |
| N         | horizon length                     | 40                     |
| \(\Delta t\) | discretization step               | 0.2 s                  |
| \(p_{M,\text{ref}}\) | reference position of free ball    | \([7.5, 0, 0]\)^T m    |

multiple shooting discretization of ODE [7]:

\[
\begin{align*}
\minimize_{u_0, \ldots, u_{N-1}} & \quad \sum_{k=0}^{N-1} \begin{bmatrix} x_k - x_{\text{ref}} \\ u_k - u_{\text{ref}} \end{bmatrix}^T \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_k - x_{\text{ref}} \\ u_k - u_{\text{ref}} \end{bmatrix} \\
& \quad + (x_N - x_{\text{ref}})^T P (x_N - x_{\text{ref}}) \\
\text{subject to} & \quad x_0 = \pi_0, \\
& \quad x_{k+1} = \phi(x_k, u_k), \quad k = 0, \ldots, N - 1, \\
& \quad -1 \leq u_k \leq 1, \quad k = 0, \ldots, N - 1,
\end{align*}
\]

where the initial state \(\pi_0\) is the current estimate of the state vector, \(\phi : \mathbb{R}^{n_x} \times \mathbb{R}^3 \rightarrow \mathbb{R}^{n_x}\) is obtained by performing a single RK4 step of length 0.2 s on ODE [7]. Furthermore, we choose a horizon length \(N = 40\) and the weighting matrices:

\[
Q = \text{diag}(0, \ldots, 0, 2.5, 2.5, 2.5, 25, \ldots, 25),
\]

\[
P = \text{diag}(0, \ldots, 0, 10, 10, 10, 0, \ldots, 0),
\]

\[
R = \text{diag}(0.1, 0.1, 0.1),
\]

and corresponding reference values:

\[
x_{\text{ref}} = [0, \ldots, 0, 7.5, 0, 0, 0, \ldots, 0]^T,
\]

\[
u_{\text{ref}} = [0, 0, 0]^T.
\]

The design parameters are chosen as in [16] and are summarized in Table 2. Note that we did not introduce path constraints or state bounds, since these are not supported by all solvers that we compare to below.

5.1.3 Closed-loop experiments

In closed-loop, an MPC controller repeatedly (approximately) solves OCP [8]. The first control \(u_0\) is passed to the dynamic system under control and a new
initial state $x_0$ is obtained. Here, we simulate the system by using a more accurate integrator than the one in OCP (8), namely the Dormand-Prince method, as implemented in the MATLAB routine ode45.

We introduce one disturbance into the closed-loop system, similar as in [62]: in the beginning of the simulation, we start from a horizontal configuration of the chain of masses. Around the midpoint of the simulation, we override the closed loop control with a constant $u_d = [-1, 1, 1]^T$. After one second of simulation time, the controller takes over again.

We compare the following solvers with each other for this particular closed-loop setup:

- IPOPT [61]. As a solver not targeting embedded devices specifically, we use it as a baseline to compare against.
- FalcOPT [56]. A projected gradient descent method tailored for NMPC.
- VIATOC [36]. A gradient projection method for MPC that only allows linear inequality constraints.
- ACADO Code Generation tool [35]. Generates SQP-based solvers.
- GRAMPC [16]. An embedded Augmented Lagrangian-based solver.
- acados. Framework presented in the current paper.

The tuning parameters for the different solvers are listed in Table 3. This benchmarking problem and accompanying MATLAB scripts have been published online, see [57].

| Solver   | Tuning parameters                      |
|----------|----------------------------------------|
| IPOPT    | Called through CasADi, default parameters |
| FalcOPT  | Tolerance (eps): 0.1, maximum number of iterations (maxIt): 100 |
| VIATOC   | Maximum number of iterations: 20       |
| ACADO    | RTI solver, Full condensing, QP solver qpOASES |
| GRAMPC   | Parameters chosen as in [16]: max. number of augm. Lagrange iterations: 5 |
| acados   | SQP RTI solver, QP solver HPIPM, partial condensing horizon of 5 |

In order to compare the quality of the closed-loop solutions, we use the notion of distance-from-reference (DR), which is an approximation of the integrated cost along closed-loop trajectories:

$$ DR_{(\cdot),n} = \sum_{i=0}^{n} \begin{bmatrix} x_{(\cdot),i} - x_{ref} \\ u_{(\cdot),i} - u_{ref} \end{bmatrix}^T \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x_{(\cdot),i} - x_{ref} \\ u_{(\cdot),i} - u_{ref} \end{bmatrix}. $$

To compare the different solvers, we plot the relative cumulative sub-optimality (RCSO), relative to a fully converged solution, in this case, the IPOPT solution, which reads as

$$ RCSO_{(\cdot),n} = \frac{DR_{(\cdot),n} - DR_{IPOPT,n}}{DR_{IPOPT,n}}. $$
Table 4 Relative suboptimality at the end of the simulation of the hanging chain with $M = 5$ and $N = 40$. First-order methods VIATOC, GRAMPC and FalcOPT were tuned to perform similarly. The algorithms chosen in ACADO and acados are identical, hence the results are identical.

| Solver name | RCSO | IPOPT | FalcOPT | VIATOC | ACADO | GRAMPC | acados |
|-------------|------|-------|---------|--------|-------|--------|--------|
| IPOPT       | 0.00e+00 | 3.170e-01 | 4.170e-01 | 4.74e-03 | 1.01e-04 | 7.17e-02 | 1.01e-04 |

Table 5 Computation times for the closed-loop experiments on a chain of masses (cf. Figure 4)

| comp. time per iteration (ms) | median | minimum | maximum |
|-------------------------------|-------|---------|---------|
| IPOPT                         | 59.84 | 49.06   | 384.90  |
| FalcOPT                       | 4.36  | 0.44    | 11.10   |
| VIATOC                        | 5.63  | 5.27    | 6.68    |
| ACADO                         | 1.97  | 1.90    | 3.45    |
| GRAMPC                        | 1.06  | 0.81    | 1.31    |
| acados                        | 1.05  | 0.87    | 2.23    |

for $n$ going from 0 to 300, the number of time steps in our simulation. We show a comparison in Table 4. The results for ACADO and acados are exactly the same, as they implement the same real-time algorithm, with both being very close to the reference solution from IPOPT. The solvers GRAMPC, VIATOC and FalcOPT, being based on first-order methods, are further away from the IPOPT solution. These findings are consistent with previously published work by other authors, see [16].

We have a look at the computational performance along the closed-loop trajectories in Figure 4. GRAMPC, ACADO, VIATOC and acados produce consistent timings throughout the entire experiment, even when the disturbance occurs. This is a beneficial property for embedded solvers, as they often have a fixed time deadline, being part of a larger control application. GRAMPC and acados produce solutions at almost the same speed, both approximately a factor 2 faster than ACADO which is in turn a factor 2-3 faster than VIATOC. Near the equilibrium, FalcOPT takes the shortest computation time, as it is performing only a few gradient steps per iteration. IPOPT is included as a baseline for comparison to non-embedded solvers. The timings are summarized in Table 5.

Of course, an optimization solver could always trade off sub-optimality for computation time. To get the full picture, we plot both measures against each other in Figure 5. We look at relative cumulative sub-optimality over the entire length of the experiment, versus worst-case computation times. By this comparison, we see that acados and GRAMPC are on the Pareto-optimal front: although acados is a factor 1000 less suboptimal than GRAMPC, the computational cost is higher. By the median computation times, acados is faster (see Table 5).
5.2 Case study 2: Hessian regularization

In Section 2.6, we briefly mentioned the impact of Hessian regularization on SQP methods. In this case study, we compare the convergence of exact-Hessian based SQP with three different Hessian regularizations, on a simple control problem.
5.2.1 System description

We control a mass on a rod (a pendulum), balanced on a horizontally moving cart, see Figure 6. The goal is to swing up the pendulum from a stable equilibrium position, namely hanging down vertically.

The dynamics of the cart-pendulum are described by the following ordinary differential equation, where \( p, v \) are the horizontal displacement and velocity, respectively, \( \theta \) is the angle with the vertical and \( \omega \) the corresponding angular velocity:

\[
\begin{align*}
\dot{p} &= v, \\
\dot{\theta} &= \omega, \\
\dot{v} &= \frac{-ml \sin(\theta) \dot{\theta}^2 + mg \cos(\theta) \sin(\theta) + F}{M + m - m(\cos(\theta))^2}, \\
\dot{\omega} &= \frac{-ml \cos(\theta) \sin(\theta) \dot{\theta}^2 + F \cos(\theta) + (M + m)g \sin(\theta)}{l(M + m - m(\cos(\theta))^2)}.
\end{align*}
\]

We collect the states in the state vector \( s = [p, \theta, v, \omega]^T \), the control \( u \) is the horizontal force \( F \). Transcribing the continuous-time OCP with multiple shooting gives rise to the following OCP:

\[
\begin{align*}
\text{minimize} & \quad x_0, \ldots, x_N, u_0, \ldots, u_{N-1} \sum_{k=0}^{N-1} \left[ x_k^T \begin{bmatrix} Q & x_k \\ R & u_k \end{bmatrix} + x_N^T Q x_N \right] \\
\text{subject to} & \quad x_0 = x_0, \\
& \quad x_{k+1} = \phi_k(x_k, u_k), \quad k = 0, \ldots, N-1, \\
& \quad -80 \leq u_k \leq 80, \quad k = 0, \ldots, N-1,
\end{align*}
\]

where \( \phi \) is an RK4 integrator, simulating (9) over one shooting interval. The weight matrices are chosen as

\[
Q = \text{diag}([1000, 1000, 0.01, 0.01]), \quad R = 0.01.
\]
Because our aim is to swing up the pendulum, we selected strong weights on the position and angle. The other states and the control are assigned a weak penalty in order to avoid too abrupt swing-ups and to favor smooth trajectories. Note that the weighting matrices $Q$ and $R$ are tuning parameters used by the control engineer in the design process in order to obtain a desired behavior. Different choices are therefore equally valid. The initial value is $s_0 = [0, \pi, 0, 0]^T$. We choose $N = 100$ shooting intervals of length 0.01 s.

### 5.2.2 Exact-Hessian based SQP

We solve OCP (10) with SQP, where we use the exact Hessian of the Lagrangian. In the notation of (3):

$$H_k = \begin{bmatrix} Q & R \\ R & \sum_{i=0}^{n_r} \lambda_{k,i} \nabla^2_{(x,u)} \phi_i(x_k, u_k) \end{bmatrix}, \quad k = 0, \ldots, N - 1$$

$$H_N = Q,$$

where $\lambda_{k,i}$ are the Lagrange multipliers associated with the dynamic equality constraints.

In some cases, the non-convexity of the dynamic equations gives rise to an indefinite Hessian matrix. We apply the project($\cdot$) and mirror($\cdot$) regularizations, as well as the convexification method (see Section 2.4). All are implemented as modules in the acados framework.

We now compare the convergence of the SQP iterates obtained through the three different regularization methods. For each SQP variant, we start the SQP iterations from the point $w^{[0]} = 0$. The result can be seen in Figure 7. The structure-exploiting convexification converges almost twice as fast as the projection regularization, and in turn much faster than the mirroring regularization. Intuitively, this makes sense, as mirroring is ‘blocking’ directions associated with large negative eigenvalues, by introducing large positive eigenvalues in those directions. This prevents the solver from taking larger steps\(^\text{2}\). In turn, the structure-exploiting regularization is faster than merely projecting the eigenvalues on the positive definite cone, because it is redistributing convexity among all stages, and thus needs less regularization overall.

It must be said that the convexification method is quite a bit more involved than the other two regularization schemes. However, by using the optimized linear algebra routines of BLASFEO, we implemented the convexification method such that it is only slightly more expensive per iteration than the basic regularization methods, see Table 6 but much more computationally cheap overall. Thus, the Hessian convexification method allows us to perform exact-Hessian based NMPC online, with better performance than state-of-the-art methods.

---

\(^2\) This is also the approach followed by the algorithms obtained with the ACADO Code Generation tool.
5.3 Case study 3: Hardware-in-the-loop experiments for an engine control application

As a last case study, we discuss the performance of acados on an embedded platform, namely the dSPACE MicroAutoboxII [15], depicted in Figure 8. It is an industrial computing platform that is used in the car industry. It features a 900 MHz PowerPC processor (IBM PPC 750GL) with 16MB of main memory. The control application that we focus on is engine control, with the engine model as presented in [4], which we will briefly reproduce here.

Two-stage turbocharging gasoline engines are investigated to overcome the drawbacks of conventional (single-stage) turbocharging. The main advantage
they offer is a better trade-off between short transient times after load changes and a high specific power. However, the two-stage architecture puts a higher demand on the engine controller, due to the nonlinear nature with cross-couplings in the inputs and the necessity to consider constraints. NMPC has been proposed as a viable control strategy [4].

In Figure 9 a sketch of the two-stage turbocharged engine is depicted. The high-pressure (HP) stage is able to realize fast transients, the low-pressure (LP) stage produces a higher specific power, but with slower dynamics. The control challenge lies in accurately tracking the boost pressure $p_{\text{boost}}$, given the highly nonlinear coupling between both stages.

For reasons of brevity, we directly present the engine model of [4] and refer the interested reader to that work for a derivation. We model the engine with a set of semi-explicit DAEs. The differential states consist of $\Pi_{c,lp}, \Pi_{c,hp}$, the pressure ratios between input and output of the compressor in the low pressure and high pressure stage, respectively. The algebraic states are $\Pi_{t,lp}, \Pi_{t,hp}$, the pressure ratios on the turbine. The inputs are the wastegate actuation pulse-width modulated signals $u_{wg,lp}, u_{wg,lp}$, which take on values between 0% (fully open) and 100% (fully closed).
Table 7 Parameter values for the two-stage turbocharged engine model

| Parameter | Unit | Value | Parameter | Unit | Value |
|-----------|------|-------|-----------|------|-------|
| $c_1$     | $\text{min}^{-1}$ | 0.0034 | $b_{1,1}$ | $\text{min}^{-1}$ | 1.49 |
| $c_2$     | $\text{min}^{-1}$ | 0.6 | $b_{4,1}$ | $\text{min}^{-1}$ | 0.0377 |
| $c_3$     | $\text{min}^{-1}$ | 43.6 | $b_{1,2}$ | $\text{min}^{-1}$ | 6.75 |
| $c_4$     | $\text{min}^{-1}$ | 0.9 | $b_{2,2}$ | $\text{min}^{-1}$ | 4.712 |
| $c_5$     | $\text{min}^{-1}$ | 25.3 | $b_{1,1}$ | $\text{min}^{-1}$ | 0 |

The resulting DAE system reads as

\[
\dot{\Pi}_{c, \text{lp}} = c_1 \left( \Pi_{1,5}^{1.5} - \Pi_{1,25}^{1.25} \right) \sqrt{\Pi_{t, \text{lp}}^{-1.5} - \Pi_{t, \text{lp}}^{-1.75}} - c_2 n_{\text{eng}} \Pi_{c, \text{hp}} (\Pi_{c, \text{lp}}^{1.29} - \Pi_{c, \text{lp}}) \tag{11}
\]

\[
0 = \Pi_{c, \text{lp}} \Pi_{c, \text{hp}} - c_3 \frac{\Pi_{0,5}^{0.5} - \Pi_{0,25}^{0.25}}{n_{\text{eng}}} \left( \sqrt{\Pi_{t, \text{lp}}^{-1.5} + c_4 A(\Pi_{c, \text{lp}} \cdot \Pi_{c, \text{hp}}, u_{\text{wg, lp}}) \right) \tag{12}
\]

\[
\dot{\Pi}_{c, \text{hp}} = c_5 \left( \Pi_{1,5}^{1.5} - \Pi_{1,25}^{1.25} \right) \sqrt{\Pi_{t, \text{hp}}^{-1.5} - \Pi_{t, \text{hp}}^{-1.75}} - c_6 n_{\text{eng}} \Pi_{c, \text{lp}} (\Pi_{c, \text{hp}}^{1.29} - \Pi_{c, \text{hp}}) \tag{13}
\]

\[
0 = \Pi_{c, \text{lp}} \Pi_{c, \text{hp}} - c_7 \frac{\Pi_{0,5}^{0.5} - \Pi_{0,25}^{0.25}}{n_{\text{eng}}} \left( \sqrt{\Pi_{t, \text{hp}}^{-1.5} + c_8 (1 - u_{\text{wg, hp}}/100) \right), \tag{14}
\]

with, additionally, $n_{\text{eng}} = 2000 \text{min}^{-1}$ the engine speed. We model it as a measured disturbance, in this case a constant. Furthermore, $A : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is defined by

\[
A(u, v) = \gamma_1(u) \cdot \gamma_2(v),
\]

with $\gamma_i : \mathbb{R} \rightarrow \mathbb{R}$:

\[
\gamma_i(u) = b_{1,i} + b_{2,i} \left( 1 + e^{-u_{a+b_3,i}} \right)^{-1}.
\]

The values of all model parameters can be found in Table 7.

In order to obtain a smooth control behavior, we include the time derivative of the controls in the optimization formulation, as follows: $u_{\text{wg, lp}} = d_{u, \text{lp}}$, $\dot{u}_{\text{wg, hp}} = d_{u, \text{hp}}$, and we collect these rates in

\[
d = \begin{bmatrix} d_{u, \text{lp}} \\ d_{u, \text{hp}} \end{bmatrix}.
\]
We then define the vector of differential states, algebraic states and controls, respectively, as follows:

\[
x = \begin{bmatrix} \Pi_{c,lp} \\ \Pi_{c,hp} \\ u_{wg,lp} \\ u_{wg,hp} \end{bmatrix}, \quad z = \begin{bmatrix} \Pi_{t,lp} \\ \Pi_{t,hp} \end{bmatrix}, \quad u = \begin{bmatrix} d_{u,lp} \end{bmatrix}.
\]

The control objective is to track a boost pressure signal, where the boost pressure is given by \( y_p(x) := \Pi_{c,lp} \cdot \Pi_{c,hp} \). To this end, we solve an OCP arising from a multiple shooting formulation with the Gauss-Legendre method of order 6 with sampling time 0.05 s and \( N = 20 \) shooting intervals. The DAE simulation functions are denoted by \( \phi \). Let \( r(x, u) = [y_p(x); x; u] \) and \( r_N(x) = [y_p(x); x] \). The OCP then reads as

\[
\begin{align*}
\minimize & \sum_{k=0}^{N-1} \|r(x_k, u_k) - y_r\|^2_W + \|r_N(x_N) - y_r,N\|^2_{W_N} \\
\text{subject to} & \quad x_0 = x_0, \\
& \quad \begin{bmatrix} x_{k+1} \\ z_k \end{bmatrix} = \phi(x_k, u_k), \quad k = 0, \ldots, N - 1, \\
& \quad 0 \leq u_k \leq 100, \quad k = 0, \ldots, N - 1, \\
& \quad 0.5 \leq \Pi_{c,lp,k} \leq 1.757, \quad k = 1, \ldots, N, \\
& \quad 0.5 \leq \Pi_{c,hp,k} \leq 2.125, \quad k = 1, \ldots, N, \\
\end{align*}
\]

(19)

with

\[
\begin{align*}
y_r &= [y_{p,r}; 1.14; 1.54; 50; 50; 0; 0], \\
W &= \text{diag}([10^3, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-4}, 10^{-4}]), \\
y_{r,N} &= [y_{p,r}; 1.14; 1.54; 50; 50], \\
W_N &= \text{diag}([10^3, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}]).
\end{align*}
\]

Constraints on \( \Pi_{c,lp}, \Pi_{c,hp} \) have been added for component protection of the compressor.

We repeatedly solve this OCP approximately by performing real time iterations. As an underlying QP solver, we use HPIPM. When run in closed loop on the dSPACE MicroAutoboxII, the results can be seen in Figure 10. Control bounds and state bounds become active at some point in the simulation, for the high-pressure stage. The reference is tracked closely and without oscillations, which have been observed when linear-quadratic MPC is used [4]. As for the computation times, it is interesting to note that there are spikes everywhere where a jump occurs or a constraint becomes (in)active. The computation times close to the solution (i.e. at the beginning of the simulation) drop to almost zero. In any case, the maximum computation time remains under 10 ms, which is 5x faster than the sampling time of the system (50 ms).
Fig. 10 Closed-loop simulation of the engine control task with steps in the reference boost pressure. Simulations are carried out on the dSPACE MicroAutoboxII platform at a clock speed of 900 MHz.

We remark that the computation times obtained with the dSPACE MicroAutoboxII, for this HIL experiment, are about three times slower than a desktop computer with a 2.5GHz Intel Core i7-4870HQ processor.

6 Conclusion and outlook

In this article, we presented acados, a new software package for embedded optimization. It is free and open-source software that facilitates rapid testing and deployment of (N)MPC algorithms on embedded hardware platforms. For ease of use, we offer interfaces with higher-level languages such as MATLAB and Python.

Among many features that state-of-the-art NMPC algorithms require, a couple of new features that are not present in any other software package is the convexification procedure of Section 2.4, allowing the use of exact-Hessian based SQP methods in real-time, and the SCQP Hessian approximation. Additionally, the structure exploiting GNSF-IRK integrator has the potential to speed up the simulation and sensitivity propagation tasks within an NMPC scheme. Furthermore, acados features partial condensing, different state and control dimensions per multiple shooting stage, the use of BLASFEO as a linear algebra backend and facilities for using CasADi as a modeling language.
The software is shown to be embeddable, by numerical experiments on the dSPACE MicroAutoBoxII industrial computer, resulting in computation times in the millisecond range for a non-trivial NMPC problem. Furthermore, it is shown to be fast, by comparison to other embedded optimization packages.

acados is an ongoing endeavor. Future work includes extending interoperability with Simulink for easier deployment on embedded systems, and adding features for nonlinear interior-point methods, as well as other SQP-based methods like multi-level iterations.

References

1. HPIPM, High-performance interior-point qp solvers. https://github.com/giaf/hpipm (2017)
2. Albersmeyer, J.: Adjoint-based algorithms and numerical methods for sensitivity generation and optimization of large scale dynamic systems. Ph.D. thesis, University of Heidelberg (2010)
3. Albersmeyer, J., Diehl, M.: The lifted Newton method and its application in optimization. SIAM Journal on Optimization 20(3), 1655–1684 (2010)
4. Albin, T., Ritter, D., Libera, N., Quirynen, R., Diehl, M.: In-vehicle realization of nonlinear MPC for gasoline two-stage turbocharging airpath control. IEEE Transactions on Control Systems Technology pp. 1–13 (2017)
5. Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., Diehl, M.: CasADi: a software framework for nonlinear optimization and optimal control. Mathematical Programming Computation (2018)
6. Axehill, D.: Controlling the level of sparsity in MPC. Systems & Control Letters 76, 1–7 (2015)
7. Bemporad, A., Borrelli, F., Morari, M.: The explicit solution of constrained LP-based receding horizon control. In: Proceedings of the IEEE Conference on Decision and Control (CDC). Sydney, Australia (1999)
8. Bock, H.: Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen, Bonner Mathematische Schriften, vol. 183. Universität Bonn, Bonn (1987)
9. Bock, H.G.: Recent advances in parameter identification techniques for ODE. In: Numerical Treatment of Inverse Problems in Differential and Integral Equations, pp. 95–121. Birkhäuser (1983)
10. Diehl, M., Bock, H.G., Schlöder, J.P., Findeisen, R., Nagy, Z., Allgöwer, F.: Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. Journal of Process Control 12(4), 577–585 (2002)
11. Diehl, M., Ferreau, H.J., Haverbeke, N.: Efficient numerical methods for nonlinear MPC and moving horizon estimation. In: L. Magni, M. Raimondo, F. Allgöwer (eds.) Nonlinear model predictive control. Lecture Notes in Control and Information Sciences, vol. 384, pp. 391–417. Springer (2009)
12. Domahidi, A., Zgraggen, A., Zeilinger, M.N., Morari, M., Jones, C.N.: Efficient interior point methods for multistage problems arising in receding horizon control. In: Proceedings of the IEEE Conference on Decision and Control (CDC), pp. 668–674. Maui, HI, USA (2012)
13. dSPACE: Homepage. http://www.dspace.com (2006)
16. Englert, T., Völe, A., Mesmer, F., Rhein, S., Graichen, K.: A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (grampc). ArXiv e-prints (2018)
17. Ferreau, H., Lorini, G., Diehl, M.: Fast nonlinear model predictive control of gasoline engines. In: Proceedings of the IEEE International Conference on Control Applications, Munich, pp. 2754–2759 (2006)
18. Ferreau, H.J., Almer, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J.L., Stathopoulos, G., Jones, C.: Embedded optimization methods for industrial automatic control. In: Proceedings of the IFAC World Congress (2017)
19. Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., Diehl, M.: qpOASES: a parametric active-set algorithm for quadratic programming. Mathematical Programming Computation 6(4), 327–363 (2014)
20. Franke, R., Arnold, E.: Computer Intensive Methods in Control and Signal Processing, chap. Applying new numerical algorithms to the solution of discrete-time optimal control problems, pp. 105–117. Springer (1997)
21. Frasch, J.V., Sager, S., Diehl, M.: A parallel quadratic programming method for dynamic optimization problems. Mathematical Programming Computations 7(3), 289–329 (2015)
22. Frey, J., Quirynen, R., Kouzoupis, D., Pris, G., Geisler, J., Schild, A., Diehl, M.: Detecting and exploiting Generalized Nonlinear Static Feedback structures in DAE systems for MPC. In: Proceedings of the European Control Conference (ECC) (2019)
23. Frison, G.: Numerical methods for model predictive control. Master’s thesis, Department of Informatics and Mathematical Modelling, Technical University of Denmark (2012)
24. Frison, G.: Algorithms and methods for high-performance model predictive control. Ph.D. thesis, Technical University of Denmark (DTU) (2015)
25. Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A., Diehl, M.: BLASFEO: Basic linear algebra subroutines for embedded optimization. ACM Transactions on Mathematical Software (TOMS) 44(4), 42:1–42:30 (2018)
26. Frison, G., Quirynen, R., Zanelli, A., Diehl, M., Jørgensen, J.B.: Hardware tailored linear algebra for implicit integrators in embedded NMPC. In: Proceedings of the IFAC World Congress (2017)
27. Frison, G., Sørensen, H.B., Dammann, B., Jørgensen, J.B.: High-performance small-scale solvers for linear model predictive control. In: Proceedings of the European Control Conference (ECC), pp. 128–133 (2014)
28. Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Transactions on Mathematical Software 29(1), 58–81 (2003)
29. Griewank, A.: Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation. No. 19 in Frontiers in Appl. Math. SIAM, Philadelphia (2000)
30. Gros, S., Zanon, M., Quirynen, R., Bemporad, A., Diehl, M.: From linear to nonlinear MPC: bridging the gap via the real-time iteration. International Journal of Control (2016)
31. Grüne, L., Pannek, J.: Nonlinear Model Predictive Control, second edition edn. Springer-Verlag (2017)
32. Hairer, E., Norsett, S., Wanner, G.: Solving Ordinary Differential Equations I, 2nd edn. Springer Series in Computational Mathematics. Springer, Berlin (1993)
33. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems, 2nd edn. Springer, Berlin Heidelberg (1991)
34. Herceg, M., Kvasnica, M., Jones, C., Morari, M.: Multi-Parametric Toolbox 3.0. In: Proc. of the European Control Conference, pp. 502–510. Zürich, Switzerland (2013). https://control.ee.ethz.ch/~mpt
35. Houska, B., Ferreau, H.J., Diehl, M.: An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. Automatica 47(10), 2279–2285 (2011)
36. Kalmar, J., Backman, J., Visala, A.: A toolkit for nonlinear model predictive control using gradient projection and code generation. Control Engineering Practice 39, 56–66 (2015)
37. Kouzoupis, D., Ferreau, H.J., Peyrl, H., Diehl, M.: First-order methods in embedded nonlinear model predictive control. In: Proceedings of the European Control Conference (ECC), pp. 2617–2622 (2015)
38. Kouzoupis, D., Quirynen, R., Freasch, J.V., Diehl, M.: Block condensing for fast nonlinear MPC with the dual Newton strategy. In: Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC), vol. 48, pp. 26–31 (2015)
39. Kraus, T., Ferreau, H.J., Kayacan, E., Ramon, H., Baedemacker, J.D., Diehl, M., Saeys, W.: Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles. Computers and Electronics in Agriculture 98, 25–33 (2013)
40. Li, W.C., Biegler, L.T.: Multistep, Newton-type control strategies for constrained nonlinear processes. Chem. Eng. Res. Des. 67, 562–577 (1989)
41. Liniger, A., Domahidi, A., Morari, M.: Optimization-based autonomous racing of 1:43 scale RC cars. Optimal Control Applications and Methods 36(5), 628–647 (2015)
42. Maciejowski, J.M.: Predictive Control with Constraints. Prentice Hall (2002)
43. Nocedal, J., Wright, S.J.: Numerical Optimization, 2 edn. Springer Series in Operations Research and Financial Engineering. Springer (2006)
44. Ohtsuka, T.: A continuation/GMRES method for fast computation of nonlinear receding horizon control. Automatica 40(4), 563–574 (2004)
45. Qin, S., Badgwell, T.: An overview of industrial model predictive control technology. In: J. Kantor, C. Garcia, B. Carnahan (eds.) Fifth International Conference on Chemical Process Control – CPC V, pp. 232–256. American Institute of Chemical Engineers (1996)
46. Quirynen, R.: Numerical simulation methods for embedded optimization. Ph.D. thesis, KU Leuven and University of Freiburg (2017)
47. Quirynen, R., Gros, S., Diehl, M.: Efficient NMPC for nonlinear models with linear subsystems. In: Proceedings of the IEEE Conference on Decision and Control (CDC), pp. 5101–5106 (2013)
48. Quirynen, R., Gros, S., Diehl, M.: Inexact Newton-type optimization with iterated sensitivities. SIAM Journal on Optimization 28(1), 74–95 (2018)
49. Quirynen, R., Gros, S., Houska, B., Diehl, M.: Lifted collocation integrators for direct optimal control in ACADO toolkit. Mathematical Programming Computation 9(4), 527–571 (2017)
50. Quirynen, R., Houska, B., Diehl, M.: Efficient symmetric Hessian propagation for direct optimal control. Journal of Process Control 50, 19–28 (2017)
51. Rao, C.V., Wright, S.J., Rawlings, J.B.: Application of interior-point methods to model predictive control. Journal of Optimization Theory and Applications 99, 723–757 (1998)
52. Rawlings, J.B., Mayne, D.Q., Diehl, M.M.: Model Predictive Control: Theory, Computation, and Design, 2nd edition edn. Nob Hill (2017)
53. Steinbach, M.: Fast recursive SQP methods for large-scale optimal control problems. PhD thesis, University of Heidelberg, IWR (1995)
54. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: An operator splitting solver for quadratic programs. ArXiv e-prints (2017)
55. Torrisi, G., Frick, D., Robbiani, T., Grammatico, S., Smith, R., Morari, M.: FalcOpt: First-order Algorithm via Linearization of Constraints for OPTimization. https://github.com/torrisig/FalcOpt (2017)
56. Verschueren, R., Grammatico, S., Smith, R., Morari, M.: A projected gradient and constraint linearlinear method for nonlinear model predictive control. ArXiv e-prints (2016)
57. Verschueren, R.: embench – benchmarking suite for embedded optimization. https://github.com/roversch/embench (2018)
58. Verschueren, R., van Duijkeren, N., Quirynen, R., Diehl, M.: Exploiting convexity in direct optimal control: a sequential convex quadratic programming method. In: Proceedings of the IEEE Conference on Decision and Control (CDC) (2016)
59. Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Quirynen, R., Diehl, M.: Towards a modular software package for embedded optimization. In: Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC) (2018)
60. Verschueren, R., Zanon, M., Quirynen, R., Diehl, M.: A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control. SIAM Journal of Optimization 27(3), 2085–2109 (2017)
61. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1), 25–57 (2006)
62. Wirsching, L., Bock, H.G., Diehl, M.: Fast NMPC of a chain of masses connected by springs. In: Proceedings of the IEEE International Conference on Control Applications, Munich, pp. 591–596 (2006)

63. Zanelli, A., Donahidi, A., Jerez, J.L., Morari, M.: FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. International Journal of Control (2017)

64. Zanelli, A., Horn, G., Frison, G., Diehl, M.: Nonlinear model predictive control of a human-sized quadrotor. In: Proceedings of the European Control Conference (ECC), pp. 1542–1547 (2018)

65. Zanelli, A., Quirynen, R., Diehl, M.: Efficient zero-order nmpc with feasibility and stability guarantees. In: Proceedings of the European Control Conference (ECC), Naples, Italy (2019)